A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference

Marc Toussaint

September 27, 2016

Abstract. Many state-of-the-art approaches to trajectory optimization and optimal control are intimately related to standard Newton methods. For researchers that work in the intersections of machine learning, robotics, control, and optimization, such relations are highly relevant but sometimes hard to see across disciplines, due also to the different notations and conventions used in the disciplines. The aim of this tutorial is to introduce to constrained trajectory optimization in a manner that allows us to establish these relations. We consider a basic but general formalization of the problem and discuss the structure of Newton steps in this setting. The computation of Newton steps can then be related to dynamic programming, establishing relations to DDP, iLQG, and AICO. We can also clarify how inverting a banded symmetric matrix is related to dynamic programming as well as message passing in Markov chains and factor graphs. Further, for a machine learner, path optimization and Gaussian Processes seem intuitively related problems. We establish such a relation and show how to solve a Gaussian Process-regularized path optimization problem efficiently. Further topics include how to derive an optimal controller around the path, model predictive control in constrained k-order control processes, and the pullback metric interpretation of the Gauss-Newton approximation.

1 Introduction

It is hard to track down explicitly when Newton methods were first used for trajectory optimization. As the method is centuries old it seems fair to assume that they were used from the very beginning. More recent surveys, such as (Betts, 1998; Von Stryk and Bulirsch, 1992), take Newton methods and standard non-linear constrained mathematical programming (NLP) methods as granted. Betts (1998) for instance states that Newton methods were the standard in the 60's, often executed analytically by hand. Presumably the Apollo missions relied on Newton methods to compute paths. In the 70's, with raising computational powers and quasi-Newton methods (such as BFGS), they became prevalent for many kinds of control problems.

Why do we need, half a century later, a tutorial on Newton methods for trajectory optimization? Especially in the last decade the fields of machine learning, AI, robotics, optimization and control became more and more intertwined, with methods of one discipline fertilizing ideas or complementing methods in another. This often leads to great advances in the fields. However, the interrelations between methods in the different fields are sometimes hard to see and acknowledge because the languages differs, textbooks are not cross-disciplinary, and technical papers cannot focus on length on this.

Many interesting novel approaches to trajectory optimization have been proposed in the last decade. However, identifying and relating the actual state-of-the-art across disciplines is hard. An excellent and very necessary paper in the robotics community (TrajOpt; Schulman et al., 2013), proposing non-linear mathematical programming (NLP) for trajectory optimization, might in other communities perhaps have been located decades earlier. That paper is in fact an important answer on previous papers within robotics, esp. (CHOMP; Ratliff et al., 2009), that have not compared to the NLP view on trajectory optimization. To comment also on own work, the Approximate Inference approach to Trajectory Optimization (AICO; Toussaint, 2009a) establishes important relations between iterative message passing and trajectory optimization (see below) and still inspires great advances in the field (Dong et al., 2016). But the optimization view on the same problem formulation leads to basic Newton methods that can more easily be extended to hard constraints and are more robust in practice. Similarly, it seems important to acknowledge the tight relations between the optimal control approaches DDP (Mayne, 1966) and iLQG (Todorov and Li, 2005) and plain (Gauss-) Newton methods, as discussed in more detail below.

In this tutorial we take the stand that such methods and especially their relations are best understood by considering optimization as their common underlying foundation, in particular the Newton method. With this we hope to give a basis for fertilization and understanding across disciplines.

What is proposed in this tutorial is not fundamentally novel: We discuss basic Newton and NLP methods for a general problem formulation, including also control and model predictive control around the optimum. However, some specifics of the presentation are novel, for instance:

(i) The specific k-order path optimization formulation is in contrast to the more common phase-space formulation of path problems. This, and the banded problem Jacobians and Hessians were previously mentioned in (Toussaint, 2014b).
(ii) The particular generalization of dynamic programming and model-predictive control to constrained k-order processes are, to our knowledge, novel. Also the related *approximate constrained* Linear-Quadratic Regulator (acLQR) around an optimal path has, to our knowledge, not been described in this form before. Related work is (Tassa et al., 2014).
(iii) The intimate relations between Newton-based trajectory optimization and Graph-SLAM have only very recently been mentioned (Dong et al., 2016); the recast of CHOMP as plain Newton that drops some terms seems novel.
(iv) Dong et al. (2016) also introduced the interesting idea to consider global-scale Gaussian Process smoothness priors over paths and utilize GTSAM to optimize the resulting problem. Here we propose a simpler approach to account for "banded-support" covariance kernels in the path objective with leads to linear-in-T complexity of computing Newton steps.
(v) Throughout the paper we discuss complexities of computing the Newton steps, which has not been presented in this way before.

1.1 Structure of this tutorial

Although the material presented is closely related to optimal control, we think it is insightful for this tutorial to first consider a pure trajectory optimization perspective. Controls and optimal control are not mentioned until Section 4. With this we aim to show how much we can learn about the structure of Newton-based path optimization that then relates intimately to optimal control methods.

Hence, in the first part, we formulate a path optimization problem of a particular k-order structure. Section 2.3 discusses the resulting banded structures of the problem Jacobian and Hessian and based on this derives the complexities of computing Newton steps. These basic properties of the Jacobian, Hessian and the computation of Newton steps seem technical, but they are the core to understand the relations discussed later. For instance, this allows us to understand relations to the pullback of Riemannian metrics in differential geometry, to Graph-SLAM methods, and to the CHOMP optimization method.

Section 3 asks how we can incorporate a more global smoothness objective in the optimization formulation. We briefly consider a B-spline representation of paths, which are intuitively very promising to enforce smoothness and speed up optimization. However, in practice they hardly reduce the number of Newton steps needed and the complexity of each Newton step is equal to non-spline representations. We then consider an alternative way to include more global smoothness objectives: with a covariance kernel function as in Gaussian Processes (GPs), efficiently optimizing the neg-log probability of a GP with a banded kernel function.

Section 4 then reconsiders the problem from an optimal control perspective. We first briefly introduce the basic optimal control framework and discuss direct vs. indirect approaches. To tackle our specific k-order path optimization problem we then consider dynamic programming to compute cost-to-go functions under hard constraints and the respective approximate constrained Linear Quadratic Regulator, which, just for sanity, is shown to be equivalent to the Riccati equation in the unconstrained LQR case. We extend the dynamic programming formulation to a model predictive control (MPC) formulation (in fact, a constrained k-order version of MPC) that allows to control around pre-optimized trajectories. Moving to the probabilistic setting the relations to DDP, iLQG and AICO become clear. On the conceptual level, this section establishes the relations between (i) inverting a banded Hessian (in a Newton step), (ii) dynamic programming and (iii) probabilistic message passing, all three of them making the linear-in-T complexity of computing Newton steps explicit.

2 k-order Path Optimization and its Structure

2.1 Problem formulation: k-order constrained path optimization (KOMO)

Let $x \in \mathbb{R}^{T \times n}$ be the path[1] of T time steps in an n-dimensional *configuration* space X. That is, in the dynamic case, x_t does not include velocities and x is not a *state* space (or phase space) trajectory. Instead, x only represents a series of configurations.

[1]We use the words *path* and *trajectory* interchangeably: we always think of a path as a mapping $[0, T] \to \mathbb{R}^n$, including its temporal profile.

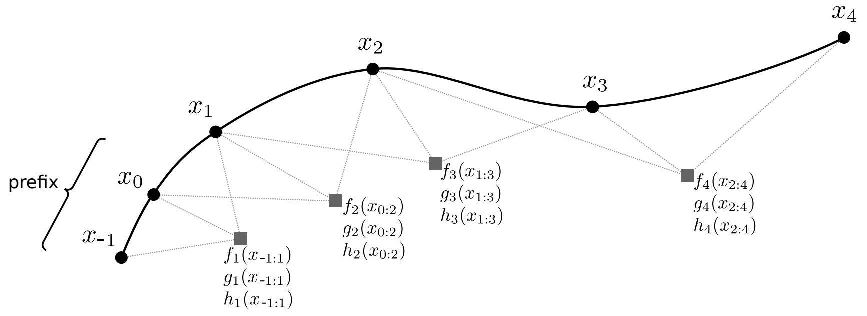

Figure 1: Illustration of the structure implied by the k-order Markov Assumption (Eq. 2)

A general non-linear program over a path x is of the form

$$\min_x f(x) \quad \text{s.t.} \quad g(x) \le 0 \;, \quad h(x) = 0 \;, \tag{1}$$

where $f : \mathbb{R}^{T\times n} \to \mathbb{R}$ is a scalar objective function, $g : \mathbb{R}^{T\times n} \to \mathbb{R}^{d_g}$ defines d_g inequality constraint functions, and $h : \mathbb{R}^{T\times n} \to \mathbb{R}^{d_h}$ defines d_h equality constraint functions. We generally assume f, g, and h to be smooth, but not necessarily convex or unimodal.

For the case of path optimization we make the following assumption:
Assumption 1 (k-order Markov Assumption). We assume

$$f(x) = \sum_{t=1}^T f_t(x_{t-k:t}) \;, \quad g(x) = \bigotimes_{t=1}^T g_t(x_{t-k:t}) \;, \quad h(x) = \bigotimes_{t=1}^T h_t(x_{t-k:t}) \;, \tag{2}$$

for a given *prefix* $x_{k\text{-}1:0}$, where each f_t is scalar, g_t is d_{gt}-dimensional, and h_t is d_{ht}-dimensional.

Here we use the tuple notation $x_{t-k:t} = (x_{t-k}, x_{t-k+1}, .., x_t)$. The prefix $x_{k\text{-}1:0}$ are the robot configurations before the path; assuming this to be known simplifies the notation, without the need to introduce a special notation for the first k terms. The outer product $\bigotimes$ notation means that the constraint functions g_t of each time step are stacked to become the full $(d_g = \sum_t d_{gt})$-dimensional constraint function g over the full path. Under this assumption, we define our problem as

Definition 1 (k-order Motion Optimization (KOMO; Toussaint, 2014b))**.**

$$\min_x \sum_{t=1}^T f_t(x_{t-k:t}) \quad \text{s.t.} \quad \forall_{t=1}^T : \;\; g_t(x_{t-k:t}) \le 0 \;, \;\; h_t(x_{t-k:t}) = 0 \;. \tag{3}$$

Figure 1 illustrates the structure implied by the k-order Markov Assumption: Tuples $x_{t-k:t}$ of $k+1$ consecutive variables are coupled by the objectives and constraints

$$\phi_t(x_{t-k:t}) \triangleq \begin{pmatrix} f_t(x_{t-k:t}) \\ g_t(x_{t-k:t}) \\ h_t(x_{t-k:t}) \end{pmatrix} . \tag{4}$$

We call these $\phi_t(x_{t-k:t}) \in \mathbb{R}^{1+d_{gt}+d_{ht}}$ the **features** at time t, encompassing cost, inequality, and equality features. In Fig. 1, the coupling features $\phi_t(x_{t-k:t})$ are represented by the boxes. The graphical notation is used in analogy to factor graphs and conditional random fields (CRFs) (Kschischang et al., 2001; Lafferty et al., 2001), helping us to discus these relations already on the level of the problem formulation.

The structure of CRFs is typically captured in the form

$$P(y|x) = \frac{1}{Z(x,\beta)}\ \exp\{\sum_i \tilde\phi_i(y_{\partial i},x)^\top \beta_i\}\ , \tag{5}$$

where $\tilde\phi_i(y_{\partial i},x)$ are *features* that couple the input x to a tuple $y_{\partial i}$ of output variables.[2] These features capture the structure of the output distribution $P(y|x)$. Going back to path optimization, in our case the features $\phi_t(x_{t-k:t})$ not only encompass costs, but also inequality and equality constraints. As plain path optimization is not a learning problem, we have no global model parameters β. However, as a side note, in the case of inverse optimal control it is exactly the case that we want to parameterize an unknown path cost function and learn it from data—which can be done exactly by introducing parameters β that weight potential cost features, as in CRFs (Englert and Toussaint, 2015).

2.2 Background on basic constrained optimization

The field of optimization has developed a large amount of methods for non-linear programming—see (Nocedal and Wright, 2006) for an excellent introduction. These existing methods are highly relevant also in the context of path optimization. We cannot review in detail the material here. Instead we summarize, in a very subjective nutshell, a few essential insights from the field of optimization as follows:

(i) The core two issues in unconstrained optimization are *stepsize* and *step direction*.
(ii) Concerning stepsize, solid adaptation schemes with guarantees are line search, backtracking, Wolfe conditions, and trust regions.
(iii) Concerning step direction, the Newton direction is the golden standard. If Hessians are not readily available, try to approximate them (quasi-Newton methods, BFGS) or at least account for correlations of gradients or the search space metric (conjugate gradient, natural gradient). Never use plain gradients or even black-box sampling if there is a chance to be more informed towards Newton directions. The Hessian represents the structure of the problem, analogous to graphical models and factor graphs (see below)—and efficiency requires to exploit such structure.
(iv) There are various ways to address constrained programs by solving a series of unconstrained problems, in particular: log-barrier (interior point), primal-dual-Newton, augmented Lagrangian, and sequential quadratic programming (SQP). If done properly, each of these approaches might lead to comparable performance and the best choice depends on the specifics of the application. Arguably, this choice is less relevant than the previous two points.

As a consequence, in the case of path optimization we need to discuss especially the structure of the problem, that is, the structure of the Hessian. This will be a central topic of this tutorial, and we will discuss how this structure relates to factor graphs and graphical models, and how exploitation of this structure in terms of the respective linear algebra methods is analogous or equivalent to message passing or dynamic programming in such graphical models.

[2] ∂i denotes the neighborhood of feature i in the bipartite graph of features and variables; and thereby indexes the tuple of variables on which the ith feature depends.

In the case of *un*constrained optimization ($d_g = d_h = 0$), we could directly consider the structure of Newton steps

$$-\nabla^2 f(x)^{\text{-1}}\ \nabla f(x) \tag{6}$$

under our assumptions. However, as we are concerned with a constrained problem we first want to recap standard approaches to constrained optimization and discuss what the implication of these approaches is w.r.t. the structure of the resulting Newton steps. We focus on sequential quadratic programming (SQP) and the augmented Lagrangian (AuLa) method, and only briefly mention standard log barrier and primal-dual Newton methods.

The Newton method steps, in every iteration, towards the optimum of a local 2nd-order Taylor approximation of $f(x)$. Sequential Quadratic Programming (SQP, see (Nocedal and Wright, 2006) for details) is a direct generalization of this: In every iteration we step towards the optimum of a local Taylor approximation of the original constrained problem (1). Concretely, we compute the local 2nd-order Taylor of the objective,

$$f(x+\delta) \approx f(x) + \nabla f(x)^\top \delta + \frac{1}{2}\delta^\top \nabla^2 f(x) \delta\ , \tag{7}$$

and the local 1st-order Taylor of the constraints,

$$g(x+\delta) \approx g(x) + \nabla g(x)^\top \delta\ , \quad h(x+\delta) \approx h(x) + \nabla h(x)^\top \delta\ . \tag{8}$$

This defines the sub-problem

$$\min_\delta\ f(x) + \nabla f(x)^\top \delta + \frac{1}{2}\delta^\top \nabla^2 f(x) \delta \quad \text{s.t.} \quad g(x) + \nabla g(x)^\top \delta \le 0\ , \quad h(x) + \nabla h(x)^\top \delta = 0\ , \tag{9}$$

which can be solved with a standard Quadratic Programming solver. In a robotics context, the computation of the terms $\nabla f(x), \nabla^2 f(x), \nabla g(x), \nabla h(x)$ is typically expensive, requiring to *query* kinematics, dynamics and collision models; but once these terms are computed locally at x, the sub-problem of computing δ^* considers these as constant and does not require further queries. The dimensionality of the sub-problem (9) is though still the same as that of (1). As in ordinary Newton methods, the optimal δ^* only defines a good search direction and we need to backtrack until we found a point that decreases f sufficiently (Wolfe condition) *and* that is feasible—these criteria again require the real kinematics, dynamics and collision models to be queried.

As a general conclusion, an optimizer should try to reduce the number of queries as mush as possible by putting much effort in deciding on a good step direction and stepsize. SQP does so by solving the QP (9).

SQP became a standard in robotics. However, we want to also highlight another method that is not as frequently mentioned in the robotics context and not well documented for the inequality case: the augmented Lagrangian (AuLa) method (Conn et al., 1991; Toussaint, 2014a). The method is simple and effective. First consider an imprecise but common practice to handle constraints, namely by adding squared penalty terms. Instead of solving (1) we address

$$F(x) = f(x) + \nu \sum_j h_j(x)^2 + \mu \sum_i [g_i(x) > 0]\ g_i(x)^2\ , \tag{10}$$

which adds squared penalties if constraints are violated.[3] $F(x)$ can be efficiently minimized by a standard Gauss-Newton method, which approximates the Hessian of $F(x)$ by $\nabla^2 F(x) \approx \nabla^2 f(x) + \nu \sum_j \nabla h_j(x) \nabla h_j(x)^\top + \mu \sum_i [g_i(x) > 0]\ \nabla g_i(x) \nabla g_i(x)^\top$.

Because the squared penalties are flat at $h_j = 0$ and $g_i = 0$, minimizing $F(x)$ will lead to constraint violations for the critical (active) constraints. The amount of violation could be controlled by increasing ν and μ. However, there is a very elegant alternative: from the amount of violation we can guess Lagrange parameters that, in the next iteration, push out of constraint violations and "should" lead to satisfied constraints. Concretely, we define the augmented Lagrangian as

$$\hat{L}(x) = f(x) + \sum_j \kappa_j h_j(x) + \sum_i \lambda_i g_i(x) + \nu \sum_j h_j(x)^2 + \mu \sum_i [g_i(x) > 0]\ g_i(x)^2\ , \quad (11)$$

which includes both, squared penalties and Lagrange terms.

In the first iteration, $\kappa = \lambda = 0$ and $\hat{L}(x) = F(x)$. We compute $x' = \min_x \hat{L}(x)$, and then reassign Lagrange parameters using the AuLa updates[4]

$$\kappa_j \leftarrow \kappa_j + 2\nu h_j(x')\ , \quad \lambda_i \leftarrow \max(\lambda_i + 2\mu g_i(x'), 0). \quad (12)$$

Note that $2\nu h_j(x')$ is the force (gradient) of the equality penalty at x', and $\max(\lambda_i + 2\mu g_i(x'), 0)$ is the force of the inequality constraint at x'. What this update does is it considers the forces exerted by the penalties, and translates them to forces exerted by the Lagrange terms in the next iteration. This tries to trade the penalizations for the Lagrange terms. It is straightforward to prove that, if f, g and h are linear and the same constraints are active in two consecutive iterations, the AuLa update (12) assigns "correct" Lagrange parameters, all penalty terms are zero in the second iteration, and therefore the solution fulfills the first KKT condition after one iteration (Toussaint, 2014a). The convergence behavior and efficiency is, in practice, very similar to the simple and imprecise squared penalty approach, while it leads to precise constraint handling. Unlike SQP it does not need a QP solver for the sub-problems, but only a robust Gauss-Newton method on $\hat{L}(x)$. For reference, we include a basic robust Newton method in Table 1.

SQP and AuLa are excellent choices for constrained path optimization also because in practice they can be made rather robust against numerically imprecise and non-smooth objective and constraint functions. For instance, the distance between two convex 3D polyhedra is a continuous but only piece-wise smooth function; the gradients and Hessian discontinuously depend on what are the closest points on the polyhedra. Levenberg-Marquardt damping and the Wolfe conditions help to make standard Newton methods still lead to efficient monotone decrease. The log barrier method is an approach to constrained optimization that, in our experience, interferes non-robustly with such imprecisions of constraint gradients—presumably because of the extreme conditioning of the barrier functions at convergence.

Primal-dual Newton methods are an equally strong candidate for path optimization as SQP and AuLa, as they share many structural aspects. The primal and dual variables are updated conjointly using Newton steps. Thereby we can equally exploit the structure of the Hessian as we will discuss it in the following. However, for the sake of brevity we do not go into more details of primal-dual Newton methods.

[3] [*expr*] is the indicator function of a boolean expression.

[4] There is little literature on the AuLa updates to handle inequalities. The update rule described here is mentioned in by-passing in (Nocedal and Wright, 2006); a more elaborate, any-time update that does not strictly require $x' = \min_x \hat{L}(x)$ is derived in (Toussaint, 2014a), which also discusses more literature on AuLa.

```
Input: initial $x \in \mathbb{R}^n$, functions $f(x), \nabla f(x), \nabla^2 f(x)$, tolerance $\theta$, parameters (defaults:
    $\varrho_\alpha^+ = 1.2, \varrho_\alpha^- = 0.5, \varrho_\lambda^+ = 1, \varrho_\lambda^- = 0.5, \varrho_{\text{ls}} = 0.01$)
Output: $x$
 1: initialize stepsize $\alpha = 1$ and damping $\lambda = \lambda_0$
 2: repeat
 3:     compute $d$ to solve $[\nabla^2 f(x) + \lambda \mathbf{I}]\ d = -\nabla f(x)$
            if $[\nabla^2 f(x) + \lambda \mathbf{I}]$ is not positive definite, increase $\lambda \leftarrow 2\lambda - \sigma_{\text{min}}$
 4:     while $f(x + \alpha d) > f(x) + \varrho_{\text{ls}} \nabla f(x)^\top (\alpha d)$ do          // line search
 5:         $\alpha \leftarrow \varrho_\alpha^- \alpha$                                      // decrease stepsize
 6:         optionally: $\lambda \leftarrow \varrho_\lambda^+ \lambda$ and recompute $d$              // increase damping
 7:     end while
 8:     $x \leftarrow x + \alpha d$                                                // step is accepted
 9:     $\alpha \leftarrow \min\{\varrho_\alpha^+ \alpha, 1\}$                                 // increase stepsize
10:     optionally: $\lambda \leftarrow \varrho_\lambda^- \lambda$                              // decrease damping
11: until $\|\alpha d\|_\infty < \theta$
```

Table 1: A basic robust Newton method. Line 3 computes the Newton step $d = -\nabla^2 f(x)^{-1} \nabla f(x)$; in practice, e.g., use the Lapack routine `dposv` to solve $Ax = b$ using Cholesky. The parameter λ controls the Levenberg-Marquardt damping, being dual to trust region methods, and makes the parabola steeper around current x.

2.3 The structure of the Jacobian and Hessian

We can summarize the previous section by observing that AuLa requires to compute Newton steps of $\hat{L}(x)$,

$$-\Big[\nabla^2 f(x) + \nu \sum_j \nabla h_j(x) \nabla h_j(x)^\top + \mu \sum_i [g_i(x) > 0]\ \nabla g_i(x) \nabla g_i(x)^\top\Big]^{-1}$$
$$(\nabla f(x) + (\kappa + 2\nu)^\top \nabla h(x) + (\lambda + 2\mu I_{[g_i(x)>0]})^\top \nabla g(x))\ , \tag{13}$$

and SQP will apply Newton steps in one or another way to solve the sub-problem (9), which structurally will involve the same or similar terms as in (13). The efficiency of both approaches hinges on how efficiently we can compute such Newton steps, and this depends on the structure of the bracket term.

Going back to our k-order Markov Assumption (2), the Jacobian of the features

$$\phi(x) = \bigotimes_{t=1}^{T} \phi_t(x_{t-k:t})\ , \quad J(x) = \frac{\partial \phi(x)}{\partial x} \tag{14}$$

reflects the factor graph structure illustrated in Fig. 1. Namely, Fig. 2 shows that the Jacobian is composed of blocks of rows, each one corresponding to a time t, which are non-zero only for those columns that correspond to the tuple $x_{t-k:t}$. Storing the dense Jacobian would require a $Tn \times (T + d_g + d_h)$-dimensional matrix with many zeros. A more natural storage of such a matrix is a *row-shifted packing*, which clips all the leading zeros of a row (shifting them to the left) and stores the number of zeros clipped. This leads to a matrix of at most $(k+1)n$ non-zero columns. Trivially we have:

Lemma 1. *If A is a row-shifted matrix of width l, the product $A^\top A$ is a banded symmetric matrix of band width $2l - 1$.*

Proof. Let s_i be the shift (number of clipped zeros) of the ith row of A. Let $B = A^\top A$. We

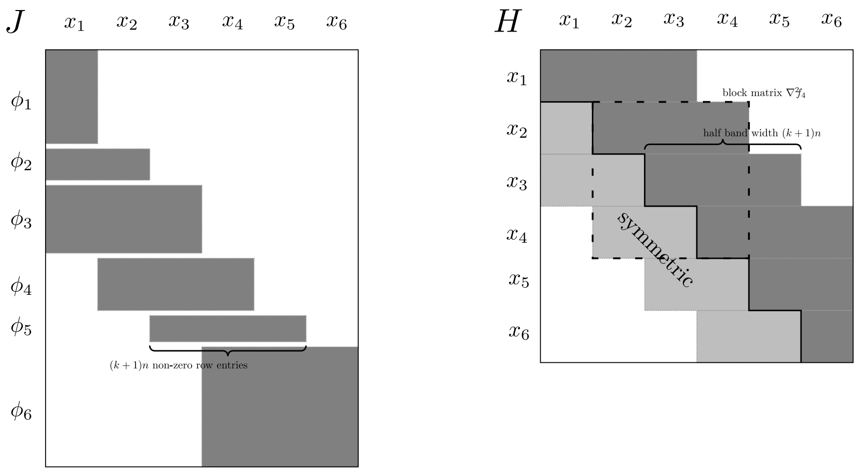

Figure 2: Structure of the Jacobian and Hessian, illustrated for $k = 2$.

have

$$B_{ij} = \sum_{t=1}^{n} A_{ti} A_{tj} = \sum_{t=1}^{n} A_{ti} A_{tj} [s_t \le i < s_t + l][s_t \le j < s_t + l] \; . \tag{15}$$

If $|i-j| \ge l$, then i and j can never be in the same interval $[s_t, s_t+l)$, and $B_{ij} = 0$. Therefore B has a band width of $2l - 1$. □

In (13) the constraints contribute to the approximate Hessian with terms $\nabla h_j(x) \nabla h_j(x)^\top$ and $\nabla g_j(x) \nabla g_j(x)^\top$. Therefore:

Corollary 2. *Under the k-order Markov Assumption, the matrix* $J(x)^\top J(x)$ *with* $J(x) = \frac{\partial \phi(x)}{\partial x}$ *is banded symmetric with width* $2(k{+}1)n - 1$.

The Hessian $\nabla^2 f(x)$ of the cost features has the structure

$$\nabla^2 f(x) = \sum_{t=1}^{T} \nabla^2 f_t(x_{t-k:t}) \; . \tag{16}$$

Each $\nabla^2 f_t(x_{t-k:t})$ is a $(k{+}1)n \times (k{+}1)n$ block matrix, as illustrated in Fig. 2. The sum of these block matrices is again banded symmetric and we have

Corollary 3. *Under the k-order Markov Assumption, the Hessian* $\nabla^2 f(x)$ *is banded symmetric with width* $2(k{+}1)n - 1$.

2.4 Computing Newton steps for banded symmetric Hessians

In the previous section we established the banded symmetric structure of the Hessian of the augmented Lagrangian. Also when using SQP or other constrained optimization approaches, the Hessian for computing Newton steps in the sub-problems will have this structure, and the efficiency of path optimization will crucially hinge on the efficiency of computing these Newton steps. Specifically, we have:

Lemma 4. *The complexity of computing Newton steps* $-A^{-1} b$ *for a banded symmetric* A *of band-width* $2l - 1$ *and* $b \in \mathbb{R}^m$ *is* $O(ml^2)$.

Proof. Golub and Van Loan (2012) describes in Section 4.3.5 explicit Algorithms for computing the Cholesky decomposition of for banded symmetric matrices (Alg. 4.3.5) with complexity $O(ml^2)$. Solving the remaining banded triangular system (Alg. 4.3.2) is $O(ml)$. □

As a side note, these algorithms are accessible in LAPACK as `dpbsv`, which internally first computes the Cholesky decomposition using `dpbtrf` and then uses `dpbtrs` to solve the remaining linear equation system.

Corollary 5. *The complexity of computing Newton steps of the form* $[\nabla^2 f(x) + J(x)^\top J(x)]^{\text{-}1} b$ *(as for the KOMO problem (3)) is* $O(Tk^2n^3)$.

We emphasize that the complexity is only linear in the number T of time steps.

2.5 Sum-of-square costs, Gauss-Newton methods, and the pullback of features space metrics

The path cost terms $f_t(x_{t-k:t})$ are, in practice, often sums-of-squares. For instance, to get smooth paths we might want to minimize squares of accelerations,

$$\|x_t + x_{t\text{-}2} - 2x_{t\text{-}1}\|^2 \ .$$

In optimal control, we typically want to minimize $\|u\|^2_H$ which, using a local approximation $u = M\ddot{q} + F$, implies cost terms

$$\|M(x_t + x_{t\text{-}2} - 2x_{t\text{-}1})/\tau^2 + F\|^2_H \ .$$

If H is Cholesky decomposed as $H = A^\top A$, this is the sum-of-squares of the features $\hat{f}_t(x_{t-k:t}) = A[M(x_t + x_{t\text{-}2} - 2x_{t\text{-}1})/\tau^2 + F]$. Given a kinematic map $\psi : \mathbb{R}^n \to \mathbb{R}^d$ (e.g., mapping to an endeffector position), we often want to penalize a squared error $\|\psi(x_t) - y^*_t\|^2_{C_t}$ to a target y_t with precision C_t. Again, with a Cholesky decomposition $C_t = A_t^\top A_t$, defining $\hat{f}_t(x_{t-k:t}) = A_t[\psi(x_t) - y^*_t]$ renders this a sum-of-squares cost.

If all cost terms are sum-of-squares of features $\hat{f}_t(x_{t-k:t})$ we have

$$\hat{f}(x) \triangleq \bigotimes_{t=1}^{T} \hat{f}_t(x_{t-k:t}) \tag{17}$$

$$f(x) = \textstyle\sum_{t=1}^{T} \hat{f}_t(x_{t-k:t})^\top \hat{f}_t(x_{t-k:t}) = \hat{f}(x)^\top \hat{f}(x) \tag{18}$$

$$\nabla f(x) = 2\nabla\hat{f}(x)^\top \hat{f}(x) \tag{19}$$

$$\nabla^2 f(x) = 2\nabla\hat{f}(x)^\top \nabla\hat{f}(x) + 2\hat{f}(x)^\top \nabla^2 \hat{f}(x) \ . \tag{20}$$

The Gauss-Newton method computes approximate Newton steps by replacing the full Hessian $\nabla^2 f(x)$ with the approximation $2\nabla\hat{f}(x)^\top \nabla\hat{f}(x)$, that is, approximating $\nabla^2 \hat{f}(x) \approx 0$. Note that the pseudo Hessian $2\nabla\hat{f}(x)^\top \nabla\hat{f}(x)$ is always semi-positive definite. Therefore, no problems arise with negative Hessian eigenvalues. The pseudo Hessian only requires the first-order derivatives of the cost features. There is no need for computationally expensive Hessians of features $\hat{f}_t$ or kinematic maps.

It is interesting to add another interpretation of the Gauss-Newton approximation, see also (Ratliff et al., 2015): The mapping $\hat{f} : \mathbb{R}^{Tn} \to \mathbb{R}^{d_f}$ maps a path to a cost feature space. We may think of both spaces as Riemannian manifolds and $\hat{f}$ a differentiable map from one manifold to the other. In the feature space, the cost $f(x)$ is just the Euclidean norm

$\hat{f}(x)^\top \hat{f}(x)$, which motivates us to think of the feature space as "flat" and define the Riemannian metric in feature space to be the Euclidean metric. Now, what is a reasonable metric to be defined on the path space? In differential geometry one defines the *pullback* of a metric w.r.t. a differentiable map $\hat{f}$ as

$$\langle x, x' \rangle_X = \left\langle d\hat{f}(x), d\hat{f}(x') \right\rangle_Y \tag{21}$$

where $d\hat{f}$ is the differential of $\hat{f}$ (a $\mathbb{R}^{d_f}$-valued 1-form) and $\langle \cdot, \cdot \rangle_Y$ is a metric in the output space of $\hat{f}$. In coordinates, and if $\langle \cdot, \cdot \rangle_Y$ is Euclidean as in our case, we have

$$\langle x, y \rangle_X = \nabla\hat{f}(x)^\top \nabla\hat{f}(x) \tag{22}$$

and therefore, *the pseudo Hessian* $2\nabla\hat{f}(x)^\top \nabla\hat{f}(x)$ *is the pullback of a Euclidean cost feature metric*. For instance, if some cost features $\hat{f}_t$ penalize velocities in feature space, finding paths x that minimize $f(x)$ corresponds to *computing geodesics* in the configuration space w.r.t. the pullback of a Euclidean feature space metric. If some cost features penalize accelerations (or control costs, as above) in some feature space, the result are geodesics in the system's phase space w.r.t. a pullback metric.

2.6 Relation to Graph-SLAM methods

Simultaneous Localization and Mapping (SLAM) is closely related to path optimization. Essentially the problem is to find a path of the camera that is consistent with the sensor readings. Graph-SLAM (Folkesson and Christensen, 2004; Thrun and Montemerlo, 2006) explicitly formulates this problem as an optimization problem on a graph.

Following the conventions of G2O (Kümmerle et al., 2011), the graph SLAM problem can be reduced to the form

$$\min_x \sum_{(i,j)\in\mathcal{C}} e(x_i, x_j, z_{ij})^\top \Omega_{ij} e(x_i, x_j, z_{ij}) \ , \tag{23}$$

where $e(x_i, x_j, z_{ij})$ is a "vector-valued error function" that indicates the consistency of states x_i and x_j with constraints z_{ij}. If we decompose the metric $\Omega_{ij} = A_{ij}^\top A_{ij}$ and define $f_{ij}(x) = A_{ij} e(x_i, x_j, z_{ij})$, this becomes a standard structured sum-of-squares problem. For $k = 1$, the KOMO problem (3) *without constraints* becomes a special case of (23), where the graph is just a chain. G2O is a highly-efficient solver for general graph least squares problems.

GTSAM (Dellaert, 2012) is another solver that allows for higher-order tuples of factors. It adopts a probabilistic interpretation of the problem (as also discussed below), but targets at computing the maximum-likelihood assignment of all random variables, which is equivalent to optimization on a factor graph. Again, unconstrained KOMO is the special k-order Markov case for such general least squares problems. Dong et al. (2016) exploit exactly these relations. They demonstrate the efficiency of using GTSAM for motion optimization, in addition to making the relation to Gaussian Processes (see below). As the approach fully exploits the structure of the problem's Hessian, the method is drastically more efficient as compared to other methods.

As a final note, none of the above consider hard constraints as we have them in KOMO. However, using, e.g., the AuLa methods it should not be hard to extend them to include hard constraints.

2.7 Relation to CHOMP

Let me briefly recap the notion of a *covariant* gradient of an objective function $f(x)$. The plain partial derivative $\nabla f(x)$ is, strictly speaking, not a vector, but a co-vector. The direction of $\nabla f(x)$ depends on the choice of coordinates. Related to this, $\nabla f(x)$ only describes the *steepest descent direction* w.r.t. a Euclidean metric. In general, the steepest descent direction should be defined depending on the metric as

$$\delta^* = \operatorname*{argmin}_\delta \nabla f(x)^\top \delta \quad \text{s.t.} \quad \langle \delta, \delta \rangle = 1 \ .$$

Here we take a step *of length one* and check how much $f(x)$ decreases in its linear approximation. The "*length one*" depends on the metric $\langle \cdot, \cdot \rangle$. If, in given coordinates, the metric is $\langle x, y \rangle = x^\top G y$, with metric tensor G, then one can show that

$$\delta^* \propto -G^{\text{-}1} \nabla f(x) \ . \tag{24}$$

It turns our that δ^* is a proper (covariant) vector that does not depend on the choice of coordinates. $\delta^*(x)$ is a *covariant* gradient of $f(x)$, more precisely, it is the covariant gradient w.r.t. the metric G. The Newton step is also a covariant vector: its direction is the covariant gradient of $f(x)$ w.r.t. the metric $H(x)$, that is, the Hessian acts as the local metric.

Covariant gradient descent therefore utilizes a metric in X to make the partial derivative become a covariant gradient. In the context of probability distributions, this metric is typically chosen to be the Fisher metric, also referred to as "natural gradient".

CHOMP (Ratliff et al., 2009) chooses the Hessian of smoothing costs as the path metric, and implements steepest descent (24) w.r.t. this metric. This is like a Newton step that drops the Hessian of the other, non-smoothing cost terms. More concretely, as smoothing cost terms CHOMP may, for instance, consider sum-of-squared accelerations $\sum_{t=1}^T f_t^2$ with cost features $\hat f_t = x_t + x_{t\text{-}2} - 2x_{t\text{-}1}$. The Hessian $H = 2\nabla \hat f^\top \nabla \hat f$ we established above is what CHOMP takes a path metric. In that sense, KOMO or any other classical Newton method generalize CHOMP to also include the Hessian of other cost terms in the Newton step.

However, this particular setting of CHOMP has the benefit that H (the Hessian of acceleration costs) is constant and sparse, making the linear algebra operations of computing quasi-Newton steps fast. Very fast kinematics and collision evaluations (using precomputed distance fields and a set-of-capsules approximation of the robot) further contributed to the performance and success of CHOMP.

3 Including more global Smoothness Objectives

Smoothness is a basic objective we have about robot motion. Typically, smoothness is implied by minimizing accelerations, control costs, or jerk along a path. While these objectives are local and comply with out local k-order Markov assumption, they still imply a form global smoothness. E.g., it is well-known that B-splines minimize squared accelerations subject to the knot constraints.

However, it is interesting to consider objectives that directly imply a form of global smoothness. We have in particular Gaussian Processes in mind, where the kernel functions directly defines the correlatedness of distal points and thereby the desired form of smoothness. We will show below that such kind of smoothness objectives are not compliant with the k-order Markov assumption, but propose ways to handle them anyway.

Before discussing Gaussian Process smoothness objectives we first consider spline encodings of paths as a means to impose global smoothness.

3.1 Splines

Basis splines, or B-splines, are a simple way to reduce the dimensionality of the path representation. First assume we want to represent a continuous 1D path $x : [0, T] \to \mathbb{R}$ with K+1 knots $z_k \in \mathbb{R}, k = 0, .., K$. For a given degree p, let $t_k \in [0, T], k = 0, .., K + p + 1$ be a series of increasing time steps associated with the knots.[5] Then we can compute coefficients[6] $b(t) \in \mathbb{R}^{K+1}$ such that $x(t) = b(t)^\top z$. Therefore, $x(t)$ is linear in the spline parameters z.

We previously defined $x \in \mathbb{R}^{T\times n}$ to be a discrete time path in n-dimensional configuration space. In this case we can compute once the discrete time spline basis matrix $\bar B \in \mathbb{R}^{T+1\times K+1}$ with $B_{t\cdot} = b(t/T)$ and then can represent

$$\bar x = \bar B \bar z \ , \tag{26}$$

with spline parameters $z \in \mathbb{R}^{K\times n}$. Here, $\bar x = x_{0:T}$ and $\bar z = z_{0:K}$ include the given start configuration $x_0 = z_0 \in \mathbb{R}^n$. To match better with the previous sections' notation we rewrite this as

$$x = Bz + bx_0^\top \ , \tag{27}$$

where $B = \bar B_{1:T,1:K}$ and $b = B_{1:T,0}$.

In conclusion, spline representations provide a simple linear re-representation of paths. In the spline representation, the feature Jacobian and Hessian are

$$J_z = J_x B \tag{28}$$
$$H_z = B^\top H_x B \ , \tag{29}$$

where J_x and H_x are the feature Jacobian and Hessian in the original path space.[7] Note that the spline basis matrix is also structured in a "banded", or row-shifted, manner, similar to the feature Jacobian. Namely,

$$b(t)_k \neq 0 \quad \Rightarrow \quad t_k \le t \le t_{k+p+1} \tag{30}$$
$$B_{tk} \neq 0 \quad \Rightarrow \quad \frac{T(k-p)}{K+1-p} \le t \le \frac{T(k+1)}{K+1-p} \tag{31}$$

[5]The time steps can, e.g., be chosen "uniformly" within $[0, T]$, $t_k = T \begin{cases} 0 & k \le p \\ 1 & k \ge K+1 \\ \frac{k-p}{K+1-p} & \text{otherwise} \end{cases}$,
which also assigns $t_{0:p} = 0$ and $t_{K+1:K+p+1} = T$, ensuring that $x_0 = z_0$ and $x_T = z_K$.

[6]The coefficients can be computed recursively. We initialize $b^0_k(t) = [t_k \le t < t_{k+1}]$ and then compute recursively for $d = 1, .., p$

$$b^d_k(t) = \frac{t - t_k}{t_{k+d} - t_k} b^{d\text{-}1}_k(t) + \frac{t_{k+d+1} - t}{t_{k+d+1} - t_{t+1}} b^{d\text{-}1}_{k\text{-}1}(t) \ , \tag{25}$$

up to the desired degree p, to get $b(t) \equiv b^p_{0:K}(t)$.

[7]As x is a matrix, J_x is, strictly speaking, a tensor and the above equations are tensor equations in which the t index of B binds to only one index of J_x and H_x.

$$\Leftrightarrow \quad (K+1-p)\, t/T - 1 \le k \le (K+1-p)\, t/T + p \,. \tag{32}$$

So the non-zero width of each row is $p+2$, and the non-zero height of each column is $T(p+2)/(K+1-p)$.

Corollary 6. *In a spline representation of degree p, the Hessian $B^\top H_x B$ has bandwidth $O(kpn)$.*

It is imperative to exploit this kind of sparsity of the spline basis matrix to ensure that the complexity of the matrix multiplication $J_x B$ in (28) is only $O(dknp)$ (recall, d is the number of features) instead of $O(dTnK)$. Equally, computing $H_x B$ in (29) is $O(Tn^2kp)$.

Now, does such a lower-dimensional spline representation of paths speed up Newton methods? We first note

Corollary 7. *The computational complexity of computing J_z is $O(dknp)$, of H_z is $O(Tn^2kp)$, of a Newton step $H_z^{-1} z$ is $O(Kn(kpn)^2)$.*

Overall, the complexity w.r.t. T is dominated by the computation of J_z and H_z and gives $O(T)$; and w.r.t. n, k it is dominated by the Newton step giving $O(k^2n^3)$. Both are exactly as for the original Newton step without spline representation. Note that also line search steps (e.g., checking the Wolfe condition) is $O(T)$ in both representations as the whole path needs to be evaluated.

If the complexity of computing Newton steps is not reduced in the spline representation, perhaps we need less iterations? We first note that Newton steps are *covariant*, that is, their direction is invariant under coordinate transforms. Therefore, if B *would* have full rank, the Newton steps are identical. Performing Newton steps on a lower-dimensional linear projection B is the same as projecting the high-dimensional Newton steps onto the low-dimensional hyperplane. There is no a priori reason for why this should lead to less iterations.

In conclusion, optimizing on a low-dimensional spline representation of the path does not necessarily lead to more efficient optimization. Empirically we often find that more Newton iterations are needed in a spline representation where the found path is less optimal.

Nevertheless, splines are a standard approach in path optimization. Perhaps the real motivation for using splines in practice is that they impose a large-scale smoothness on the solution which cannot efficiently be captured by cost features on k+1-tuples $x_{t:t+k}$. However, let us consider alternative approaches to large-scale smoothness in the following section.

3.2 Covariance smoothness objectives & Gaussian Process priors

The k-order Markov structure allows us to express smoothness objectives in terms of cost features over the kth path derivatives. Such local smoothness objectives are different to global smoothness constraints as implied by spline projections, or the kind of smoothness implied by Gaussian Process (GP) priors.

Considering the latter, for discretized time, a GP is nothing but a joint Gaussian over all path points. For instance, a GP represents the prior

$$P(x) = \mathcal{N}(x \,|\, 0, K) \propto \exp\{-\frac{1}{2} x^\top K^{-1} x\} \;, \quad K_{ts} = k(t,s) \;, \tag{33}$$

where the kernel function $k(t,s)$ is the correlation between the configurations x_t and x_s at two different times t and s. A typical kernel function used in GPs is the squared exponential

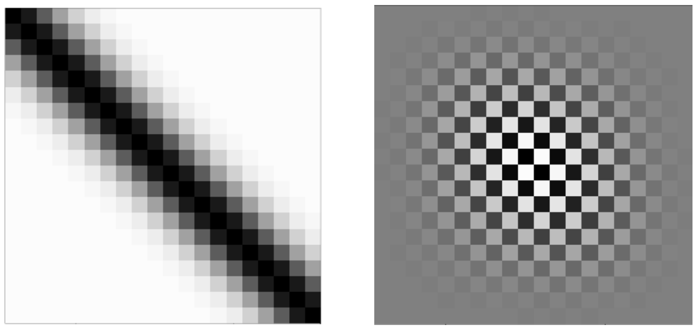

Figure 3: Left: The 20×20 covariance matrix $K_{ij} = \exp\{-((i-j)/3)^2\}$ (zero=white). Right: its inverse (precision matrix) K^{-1} (zero=gray).

kernel $k(t,s) = \exp\{-(t-s)^2/\sigma^2\}$ for some band width σ. Fig. 3(left) illustrates such a covariance matrix K in gray shading.

In our optimization context, such a GP prior translates to neg-log-probability costs, namely

$$-\log P(x) \propto \frac{1}{2} x^\top K^{-1} x \ . \tag{34}$$

Note the matrix inversion here! Fig. 3(right) illustrates the matrix K^{-1}, which turns out to be in no way 'local' or banded. This precision matrix K^{-1} plays the role of a Hessian in the cost formulation. The checker board structure can vaguely be understood as penalizing *derivatives* of the path. The rather surprising non-local structure of K^{-1} clearly breaks our k-order Markov assumption. However, it turns out that we can still compute Newton steps efficiently, in a manner that exploits the structure of K. To derive this, let us more formally define the generalized problem as

Definition 2 (Covariance regularized KOMO (CoKOMO))**.**

$$\min_x \sum_t f_t(x_{t-k:t}) + \frac{1}{2} x^\top K^{-1} x \quad \text{s.t.} \quad \forall_t : \ g_t(x_{t-k:t}) \le 0 \ , \quad h_t(x_{t-k:t}) = 0 \tag{35}$$

We define, as before, $H = \sum_t \nabla^2 f_t$ as the Hessian of the cost features, or $H = \nabla\hat{f}^\top \nabla\hat{f}$ in the Gauss-Newton case. The system's full Hessian is $H + K^{-1}$. Therefore

Corollary 8. *In CoKOMO, for a finite-support kernel, the total Hessian $\bar{H} = H + K^{-1}$ is a sum of a banded matrix H and the* inverse *of a banded matrix K.*

Computing a Newton step of the form $-\bar{H}^{-1} g$ for some g[8] can be tackled as follows

$$(H + K^{-1})^{-1} g = K(HK + I)^{-1} g \ . \tag{36}$$

Note that, if H and K are both banded, then $(HK+I)$ is banded and computing $(HK+I)^{-1}g$ is, exactly as before, $O(Tnb^2)$ if b is the bandwidth of HK. We have

Lemma 9. *If H is of semi-bandwidth h (that is, total bandwidth $2h-1$) and K is of semi-bandwidth c, then HK is of semi-bandwidth $h + c$.*

[8]In the AuLa case, $g = \nabla\hat{L}(x)$, see Eq. (12). In the SQP case, the inner loop for solving the QP (9) would compute Newton steps w.r.t. the Hessian $\bar{H}$.

Proof.

$$(HK)_{ij} = \sum_k H_{ik} K_{kj} = \sum_k [-h \le i-k \le h][-c \le j-k \le c]\; H_{ik} K_{kj} \tag{37}$$

$$= [-h-c \le i-j \le h+c]\; \sum_k H_{ik} K_{kj}\ . \tag{38}$$

□

Corollary 10. *Under the k-order Markov Assumption and including a banded covariance regularization of semi-bandwidth cn, the complexity of computing Newton steps of the form $-(H+K^{\text{-}1})^{\text{-}1} g$ is $O(T(k+c)^2 n^3)$.*

This is in comparison to the $O(Tk^2n^3)$ without the covariance regularization. We assumed a semi-bandwidth cn for K to account for the dimensionality of each $x_t \in \mathbb{R}^n$.

As a side note, the Woodbury identity and rank-one update (Sherman-Morrison formula) provide alternatives ideas to handle terms like $(H + K^{\text{-}1})^{\text{-}1}$, namely

$$(H + K^{\text{-}1})^{\text{-}1} = K - (I + KH)^{\text{-}1} KHK \tag{39}$$

$$(vv^\top + K^{\text{-}1})^{\text{-}1} = K - \frac{Kvv^\top K}{1 + v^\top K v}\ . \tag{40}$$

The first line (Woodbury) involves only banded matrices, but seems less efficient than (36). The second line (Sherman-Morrison) provides a way to recursively compute $(H+K^{\text{-}1})^{\text{-}1}$ as a series of rank-one updates if $H = \sum_i v_i v_i^\top$—as is exactly the case in the Gauss-Newton approximation $H \approx 2\nabla\hat f^\top \nabla \hat f$. Again, all computations only rely on multiplication with banded matrices.

4 The optimal control perspective

So far we have not mentioned controls at all. However, path optimization and KOMO are intimately related to standard optimal control methods. The aim of this section is two-fold, namely to clarify these relations as well as to derive algorithms for controlling a system around an optimal path.

Our starting point will be the discussion of an alternative solution approach to our optimization problem: a dynamic programming perspective on solving the general KOMO problem (3). This will be rather straight-forward, adapting Bellman's equation to the k-order constrained case, and leads to an optimal regulator around the path. This though leads to many insights:

(i) Using a 2nd-order approximation of all terms, the backward equation can be used to compute a Newton step—which now very explicitly shows the linear-in-T complexity of computing Newton steps and gives interesting insights in how the inversion of a banded symmetric matrix is related to dynamic programming on Markov chains.
(ii) Assuming a $k=1$-order linear-quadratic control process, the 2nd-order approximate backward equation coincides with the Riccati equation. This gives insights in the tight interrelations between DDP, iLQG and Newton methods.
(iii) Moving to a probabilistic interpretation of the objective function we can connect to the recent work on using probabilistic inference methods for optimal control (Rawlik et al.,

2012). In particular, backward and forward dynamic programming in our KOMO problem become equivalent to backward and forward message passing in Markov chains. Based on this we can point to the relations with path integral control methods, AICO, Ψ-learning, Expectation Maximization and eNAC that are detailed in (Rawlik et al., 2012).

4.1 Background on basic optimal control

Let us first recap the basic formulation of optimal control problems. In the discrete time setting, we consider a controlled system $x_{t+1} = f(x_t, u_t)$ and aim to minimize

$$\min_{x,u} \textstyle\sum_{t=1}^T c_t(x_t, u_t) \quad \text{s.t.} \quad x_{t+1} = f(x_t, u_t) \; . \tag{41}$$

Here we optimize over both, the state path $x = x_{1:T}$ and the control path $u = u_{1:T}$. Both are of course related by the system dynamics. Given a control path u we can compute the state path $x = F(u)$ as a function of the start state and the controls by iterating the dynamics $f(x_t, u_t)$. The control problem can be recast as

$$\min_{u} \textstyle\sum_{t=1}^T c_t(F(u)_t, u_t) \; , \tag{42}$$

and is typically solved by iteratively finding a better control path u (e.g. by a Newton step on u, or by dynamic programming, see below) and then recomputing the state path $x = F(u)$. This is called *indirect method* or multiple shooting. DDP and iLQG, which we discuss below, are such indirect methods.

This is in contrast to direct methods which instead consider x to be the optimization variable. Roughly, let $u(x_t, x_{t+1})$ be the control needed to transition from x_t to x_{t+1}. In non-holonomic systems, where not all transitions are feasible, let $h(x_t, x_{t+1}) = 0$ express an equality constraint that ensures the existence of a control signal $u(x_t, x_{t+1})$. Then the problem can be recast as

$$\min_{x} \textstyle\sum_{t=1}^T c_t(x_t, u(x_t, x_{t+1})) \quad \text{s.t.} \quad h(x_t, x_{t+1}) = 0 \; . \tag{43}$$

Such *direct* methods eliminate the controls from the problem. Our KOMO formulation is therefore a direct method.

The *dynamic programming* approach to solving such problems is to define the optimal cost-to-go function (or value function). In the indirect view (see below for the Bellman equation in the direct view) we define

$$V_t(x) = \min_{u_{t:T}} \textstyle\sum_{s=t}^T c_s(x_s, u_s) \; , \tag{44}$$

which, for every possible $x_t = x$, computes the optimal (minimal) cost for the remaining path. Bellman's optimality equation can be derived by separating out the optimization over the next control u_t,

$$\begin{aligned} V_t(x) &= \min_{u_t} \Big[c_t(x, u_t) + \min_{u_{t+1:T}} \textstyle\sum_{s=t+1}^T c_s(x_s, u_s) \Big] && (45) \\ &= \min_{u_t} \Big[c_t(x, u_t) + V_{t+1}(f(x, u_t)) \Big] \; . && (46) \end{aligned}$$

In a nutshell, the core implications of Bellman's equation are

(i) In principle we can compute all V_t recursively, iterating backward from $V_{T+1} \equiv 0$ to V_1 using equation (46). To retrieve the optimal control path u and state path x we then iterate forward

$$u_t = \underset{u_t}{\text{argmin}} \Big[c_t(x_t, u_t) + V_{t+1}(f(x, u_t))\Big] , \quad x_{t+1} = f(x_t, u_t) , \tag{47}$$

starting from the start state x_1. This forward iteration is called *shooting*. Therefore, if we can compute all V_t exactly, we can solve the optimization problem.
(ii) In the LQ case, where $f(x, u) = Ax + Bu$ is linear and $c(x, u) = x^\top Qx + u^\top Hu$ is quadratic, all cost-to-go functions are quadratic of the form $V_t(x) = x^\top \hat{V}_t x$ and the minimization in the Bellman equation (46) is analytically given by the *Riccati equation*

$$\hat{V}_t = Q + A^\top [\hat{V}_{t+1} - \hat{V}_{t+1} B(H + B^\top \hat{V}_{t+1} B)^{-1} B^\top \hat{V}_{t+1}]A , \quad \hat{V}_{T+1} = 0 . \tag{48}$$

Given we computed all $\hat{V}_t$, the optimal controls for forward shooting (47) are

$$u_t = (H + B^\top \hat{V}_{t+1} B)^{-1} B^\top \hat{V}_{t+1} A x_t , \tag{49}$$

which is call the *Linear Quadratic Regulator*. The fact that we have this optimal regulator defined globally for all possible x_t adds a fully new perspective: We can not only use it for forward shooting to find the optimal path, but we can also use it during execution as a controller to react to perturbations of our system from the planned path.
(iii) The LQ case is the analogy to the 2nd-order Taylor approximation of a non-linear objective function: To solve a non-LQ control problem one typically starts with an initial path x, approximates the dynamics and costs to 1st- or 2nd-order around x, and then solves this locally approximate problem to yield a new, better path x. There are some alternatives on how to do this in detail:

- If we approximate all terms exactly up to 2nd-order, compute all V_t in (46) and also use this second order approximation for the forward shooting (47), then this is *exactly* equivalent to a Newton step on the path optimization problem: We approximated all terms up to 2nd order and found the minimum of that local approximation. However, this is not what typical control methods do:
- If we use 2nd-order approximations to compute all V_t in (46), but then the true non-linear dynamics for forward shooting in (47), this is referred to as Differential Dynamic Programming (usually formulated in continuous time) (DDP; Mayne, 1966).
- If we use an LQ-approximation (which neglects the dynamic's Hessian) to compute all V_t using the Riccati equation, but then the true non-linear dynamics for forward shooting in (47), this is referred to as iterative LQG (iLQG; Todorov and Li, 2005).

Both, DDP and iLQG have additional details on how exactly to ensure convergence, analogous to Levenberg-Marquardt damping and backtracking in a robust Newton method. The fine difference to Newton's method has its origin in the fact that they are indirect methods, and therefore can use the exact non-linear dynamics in the forward shooting (Liao and Shoemaker, 1992). For very non-linear systems this may be beneficial (Todorov and Li, 2005).
In all three cases, the computed V_t in principle define a linear regulator around the path, which, however, does not guarantee to keep the system close to the state region where the local approximation is viable. This can be addressed using Model Predictive Control (MPC) as discussed below.

With this background, let us first discuss a (direct) dynamic programming approach to solve the KOMO problem, and then compare to standard LQG, DDP and iLQG methods.

4.2 k-order constrained Dynamic Programming and Constrained LQR Control

For easier reference we restate the general KOMO problem (3),

$$\min_x \sum_{t=1}^T f_t(x_{t-k:t}) \quad \text{s.t.} \quad \forall_{t=1}^T : \; g_t(x_{t-k:t}) \le 0 \;, \; h_t(x_{t-k:t}) = 0 \;. \tag{3}$$

Following the dynamic programming principle we define a value function over a *separator*[9] $x_{t-k:t\text{-}1}$,

Definition 3 (k-order constrained Dynamic Programming (KODP)).

$$J_t(x_{t-k:t\text{-}1}) \triangleq \min_{x_{t:T}} \sum_{s=t}^T f_s(x_{s-k:s}) \quad \text{s.t.} \quad \forall_{s=1}^T : \; g_s(x_{s-k:s}) \le 0 \;, \; h_s(x_{s-k:s}) = 0 \;, \tag{50}$$

$$= \min_{x_t} \Big[f_t(x_{t-k:t}) + J_{t\text{+}1}(x_{t-k+1:t}) \Big] \quad \text{s.t.} \quad g_t(x_{t-k:t}) \le 0 \;, \; h_t(x_{t-k:t}) = 0 \;, \tag{51}$$

$$J_{T\text{+}1} \triangleq 0 \;. \tag{52}$$

Such k-order constrained Bellman equations are comparatively rare in the literature, but straight-forward and mentioned already by Bellman in the 50's (Bellman, 1956). See also (Dohrmann and Robinett, 1999). Tassa et al. (2014) presented a DP approach for the special case with constraints on the controls only. Solving the general non-linear constrained case, computing $J_t(x_{t-k:t\text{-}1})$ for all $x_{t-k:t\text{-}1}$, is infeasible.

If, as in DDP and SQP, we approximate all cost terms f_t as second order and constraints g_t, h_t in first order, (Bemporad et al., 2002) shows an explicit derivation of an optimal constrained LQR (C-LQR) controller. The computation is complex and the resulting C-LQR is piece-wise linear and continuous, where the pieces correspond to constrained activities of the underlying QP. Bemporad et al. (2002) emphasize the benefit of computing such optimal constrained regulators offline, for all $x_{t-k:t\text{-}1}$, rather than requiring a fast local MPC within the control loop to solve the resulting QP for the current $x_{t-k:t\text{-}1}$.

An alternative approximation to the problem (50) is to not only linearize around an optimal path, but also adopt the Lagrange parameters of the optimal path (Bellman, 1956). This clearly is not optimal, as the true path might hit constraints other than the optimal path and therefore require different Lagrange parameters. But it lends itself to a simple regulator that also, using a one-step-lookahead, is guaranteed to generate feasible paths.

For *fixed* Lagrange parameters κ_t, λ_t, the dynamic programming principle for the Lagrangian is

$$\tilde J_t(x_{t-k:t\text{-}1}) \triangleq \min_{x_{t:T}} \sum_{s=t}^T f_s(x_{s-k:s}) + \lambda_s^\top g_s(x_{s-k:s}) + \kappa_s^\top h_s(x_{s-k:s}) \tag{53}$$

$$= \min_{x_t} \Big[f_t(x_{t-k:t}) + \lambda_t^\top g_t(x_{t-k:t}) + \kappa_t^\top h_t(x_{t-k:t}) + \tilde J_{t\text{+}1}(x_{t-k+1:t}) \Big] \;. \tag{54}$$

This can efficiently be computed in the LQ approximation, see below. Given $J_t(x_{t-k:t\text{-}1})$ for all t, we define

[9]We use the word separator as in Junction Trees: a separator makes the sub-trees conditionally independent. In the Markov context, the future becomes independent from the past conditional to the separator.

Definition 4 (Approximate (fixed Lagrangian) constrained LQR (acLQR)).

$$\pi_t : x_{t-k:t\text{-}1} \mapsto \underset{x_t}{\operatorname{argmin}} \Big[f_t(x_{t-k:t}) + \tilde J_{t\text{+}1}(x_{t-k+1:t})\Big]$$
$$\text{s.t.} \quad g_t(x_{t-k:t}) \le 0,\ h_t(x_{t-k:t}) = 0\ . \tag{55}$$

Note that to determine the controls at time step t, we release the Lagrange parameters again and hard constrain w.r.t. g_t and h_t. Only the Lagrange-cost-to-go function $\tilde J_{t\text{+}1}(x_{t-k+1:t})$, computed via (53), employs the fixed Lagrange parameters. If for all t a feasible x_t is found, the whole path is guaranteed to be feasible.

To compute $\tilde J_t(x_{t-k:t\text{-}1})$ in the fixed Lagrange parameter case (53), the Lagrange terms can be absorbed in the cost terms f_s. To simplify the notation let us therefore focus on the unconstrained k-order dynamic programming case,

$$\tilde J_t(x_{t-k:t\text{-}1}) = \min_{x_t} \Big[\tilde f_t(x_{t-k:t}) + \tilde J_{t\text{+}1}(x_{t-k+1:t})\Big]\ , \quad \tilde J_{T\text{+}1} = 0\ . \tag{56}$$

In the quadratic approximation we assume

$$\tilde J_t(x) = x^\top V_t x + 2 v_t^\top x + \text{const} \tag{57}$$
$$\tilde f_t(x) \approx \nabla \tilde f_t(x)^\top x + \frac{1}{2} x^\top \nabla^2 \tilde f_t(x) x + \text{const}\ . \tag{58}$$

To derive an explicit minimizer x_t in (56) we write the 2nd-order polynomial in block matrix form

$$\Big[f_t(x_{t-k:t}) + J_{t\text{+}1}(x_{t-k+1:t})\Big] \triangleq \begin{pmatrix} x_{t-k:t\text{-}1} \\ x_t \end{pmatrix}^\top \begin{pmatrix} D_t & C_t \\ C_t^\top & E_t \end{pmatrix} \begin{pmatrix} x_{t-k:t\text{-}1} \\ x_t \end{pmatrix}^\top + 2 \begin{pmatrix} d_t \\ e_t \end{pmatrix}^\top \begin{pmatrix} x_{t-k:t\text{-}1} \\ x_t \end{pmatrix} + \text{const}\ , \tag{59}$$

where the components D_t, E_t, C_t, d_t, e_t are trivially defined in terms of $\nabla^2 f_t(x), V_t, \nabla f_t(x)$, and v_t. Then

$$x_t^* = \underset{x_t}{\operatorname{argmin}} \Big[f_t(x_{t-k:t}) + J_{t\text{+}1}(x_{t-k+1:t})\Big] = -E_t^{\text{-}1}(C_t^\top x_{t-k:t\text{-}1} + e_t) \tag{60}$$
$$V_t = D_t - C_t E_t^{\text{-}1} C_t^\top\ , \quad v_t = d_t - C_t E_t^{\text{-}1} e_t\ . \tag{61}$$

To get more intuition about this equation, let us first discuss the Riccati equation as special case.

4.3 Sanity check in the LQG case and relation to DDP & iLQG

Let us assume $k = 1$ (a standard Markov chain) and standard linear control of a holonomic system,

$$x_t = A x_{t\text{-}1} + B u_{t\text{-}1}\ , \quad J_t(x_{t\text{-}1}) = \min_{x_{t:T}} \|x_t - A x_{t\text{-}1}\|^2_{\hat H} + \|x_t\|^2_Q \tag{62}$$

with $\hat H = B^{\text{-}\top} H B^{\text{-}1}$. Identifying $f_t(x_{t\text{-}1:t}) = \|x_t - A x_{t\text{-}1}\|^2_{\hat H} + \|x_t\|^2_Q$ we have

$$\nabla f_t = 2 \begin{pmatrix} -A^\top \\ 1 \end{pmatrix} \hat H (x_t - A x_{t\text{-}1}) + 2 \begin{pmatrix} 1 \\ 0 \end{pmatrix} Q x_t\ , \quad \nabla^2 f_t = 2 \begin{pmatrix} A^\top \hat H A & -A^\top \hat H \\ -\hat H A & \hat H + Q \end{pmatrix} \tag{63}$$

$$D_t = A^\top \hat{H} A \;, \quad E_t = \hat{H} + Q + V_{t+1} \;, \quad C_t = -A^\top \hat{H} \tag{64}$$

$$V_t = A^\top \Big[\hat{H} - \hat{H}(\hat{H} + Q + V_{t+1})^{-1} \hat{H} \Big] A = A^\top \Big[(\hat{H}^{-1} + \hat{V}^{-1})^{-1}) \Big] A \tag{65}$$

$$= A^\top \Big[\hat{V} - \hat{V}(\hat{H} + \hat{V})^{-1} \hat{V}) \Big] A \;. \tag{66}$$

where $\hat{V} = Q + V_{t+1}$ and the last lines use the Woodbury identity $(A^{-1} + B^{-1})^{-1} = A - A(A + B)^{-1} A$ twice. The last line is the classical Riccati equation for $\hat{V}$.

This was just a sanity check, confirming that in the unconstrained LQ-case, the DP equation (53) reduces to the standard Riccati equation. Let us recap what we have found:

(i) We know that in the unconstrained LQ case, or KOMO problem is just an unconstrained quadratic program, where the first Newton step directly jumps to the optimum.
(ii) One way to compute this Newton step (or optimum) is via the methods we described in the first part of the paper where we emphasized the importance of the structure of the Hessian as a banded symmetric matrix, allowing for the complexity $O(Tk^2n^3)$ of computing Newton steps under the KOMO assumption. We derived this complexity by looking at the respective matrix operations, in particular the implicit Cholesky decomposition.
(iii) We have now seen a second way to compute the optimum, by recursing backward the explicit DP equation (53), or (61) in the LQ approximation, which equally has complexity $O(Tk^2n^3)$. This establishes an explicit relation between matrix inversion and the dynamic programming case.
(iv) If these methods are applied to local LQ approximations of a non-linear problem, the Newton step and Riccati equation lead to the same next iterate, that is, the same next path. In that view, the standard indirect multiple shooting methods DDP and iLQG can be viewed as Newton methods that use the Riccati equation (or DDP's equation) to compute Newton steps instead of banded matrix inversion. Both algorithms also require step size controlling, such as Levenberg-Marquardt, to become robust.
(v) However, as mentioned already in Sec. 4.1, DDP and iLQG are different to Newton steps in one respect: Both use a Riccati sweep or 2nd-order Taylor approximations to compute the next *control path* u. However, the control path u is then used to compute the next *state path* $x = F(u)$ using the exact forward dynamics.
If we wanted to get equivalent iterates using Newton steps we would have to: 1) compute the next state path x using a Newton step, 2) compute the control path u for x, 3) use the exact non-linear dynamics $x' = F(u)$ to compute a corrected state path x'.

This clarifies the tight relations between classical DDP and iLQG and Newton steps in KOMO. A further technical difference is that in KOMO we can alternatively represent the problem as a $k = 2$-order process on the configuration variables, instead of as a $k = 1$-order process in phase space, which may be numerically more stable. Hard constraints have been considered in iLQG only for the special case with constraints on the controls Tassa et al. (2014). The particular k-order constrained Dynamic Programming (50) has, to our knowledge, not been proposed before.

4.4 Constrained Model Predictive Control & staying close to the reference

Stochasticity (or un-modelled additional aspects such as control delay or motor controller properties) will always lead us away from the optimal path. Depending how far we are off, the 2nd-order approximations we used to optimize the path and derive the acLQR around the path become imprecise and might lead to even more deviation from the optimal path. The standard approach to compensate for this is Model Predictive Control (MPC) (see, e.g., (Diehl et al., 2009)).

In MPC we solve, in real time, at every time step t a finite horizon trajectory optimization problem given the concrete current state x_t. This finite horizon problem will also be non-linear and require local 2nd-order approximations, but these approximations are computed at the true x_t. When an optimal path x^* was precomputed, the finite-horizon MPC problem can be defined as finding controls that steer back to the reference path, e.g., $\min_{x_{t:t+H}} \|u\|^2_H \quad \text{s.t.} \quad x_{t+H} = x^*_{t+H}$. However, MPC can also be viewed as an H-step lookahead variant of the optimal controller we get from the Bellman equation. In this view our acLQR (55) is a 1-step MPC. We can more generally define

Definition 5 (Approximate (fixed Lagrangian) constrained MPC Regulator (acMPC))**.**

$$\pi_t : x_{t-k:t\text{-}1} \mapsto \operatorname*{argmin}_{x_{t:t+H}} \Big[\sum_{s=t}^{t+H\text{-}1} f_s(x_{s-k:s}) + \tilde J_{t+H}(x_{t+H-k:t+H\text{-}1}) + \varrho \|x_{t+H} - x^*_{t+H}\|^2 \Big]$$
$$\text{s.t.} \quad \forall_{s=t}^{t+H\text{-}1} : g_s(x_{s-k:s}) \le 0,\ h_s(x_{s-k:s}) = 0 \ . \qquad (67)$$

Let's neglect the ϱ-term first. For $H = 1$ this reduces to the acLQR (55) that looks only one step ahead and relies on the (fixed Lagrangian) cost-to-go estimate $\tilde J_{t+1}$. For $H = T - t + 1$, acMPC becomes the full, long-horizon path optimization problem until termination T.

In typical applications, that is, for typical choices of the original KOMO problem (3) there is a caveat: Very often the objectives concern control costs and costs/constraints associated with the final state x_T only. The effect is that the value functions $\tilde J_{t+H}$ have, for $t \ll T$, very low eigenvalues. The resulting "gains" of the acLQR or acMPC will therefore also be very low. If the real system was linear, this would not be a problem—the Riccati equation tells us that this low-gain controller is optimal globally no matter how far we perturbed from the reference. However, in practice, for non-linear and imprecisely modeled systems, this would lead to a large and undesirable drift away from the reference, making the precomputed x^* and its local linearizations irrelevant, and be non-robust against small model errors.

The standard way to enforce staying closer to the reference during execution is to add the ϱ-term to enforce steering back to the reference at horizon H. The second option is to introduce additional penalties $\tilde f_t \leftarrow \tilde f_t + \varrho\|x_t - x^*_t\|^2$ for deviations in every time step[10] and use this $\tilde f_t$ in the backward dynamic programming (53) to compute value functions $\tilde J_t$ for the KOMO problem with cost terms $\tilde f_t$. Using such MPC we get robust trajectory tracking and can tune the stiffness of tracking by adjusting ϱ and H.

4.5 Probabilistic Interpretation and the Laplace Approximation

Let us neglect constraints and consider problems of the form

$$\min_x f(x) \ , \quad f(x) = \sum_t f_t(x_{t-k:t}) \ . \qquad (68)$$

There is a natural relation between cost (or "neg-energy", "error") and probabilities. Namely, if $f(x)$ denotes a cost for state x—or an error one assigns to choosing x—and $p(x)$ denotes

[10]Note the relation to Levenberg-Marquardt regularization

a probability for every possible value of x, then a natural relation[11] is

$$P(x) \propto e^{-f(x)} \ , \quad f(x) = -\log p(x) \ . \tag{69}$$

Given a problem of the form (68) we may therefore define a distribution over paths as

$$P(x) \propto \prod_t \exp\{-f_i(x_{t-k:t})\} \ . \tag{70}$$

It is interesting to investigate how this probability distribution over paths is related to finding the optimal path, and to stochastic optimal control under the respective costs. Note that in the optimal control context ($k = 1$), $f_t(x_{t\text{-}1:t})$ subsumes control costs and state costs, e.g., $f_t(x_{t\text{-}1:t}) = \|u\|_H^2 + \|x_t\|_R^2$ where $u = u(x_{t\text{-}1}, x_t)$.

Toussaint (2009a) and Rawlik et al. (2012) discuss an approach to stochastic optimal control that considers the distribution

$$P(x_{0:T}, u_{0:T}) \propto P(x_0) \prod_{t=0}^{T} P(x_{t+1} \,|\, x_t, u_t) \ \pi(u_t|x_t) \ \exp\{-\eta c_t(x_t, u_t)\} \ . \tag{71}$$

Here, in contrast to (70), this is the joint distribution over controls and states. Rawlik et al. (2012) discuss in detail how inference, or more precisely, minimizing KL-divergences under such probabilistic models generalizes previous approaches such as path integral control methods (Kappen et al., 2012), Approximate Inference Control (Toussaint, 2009a), but also model-free Expectation Maximization and eNAC policy search methods (Vlassis and Toussaint, 2009; Peters and Schaal, 2008).

In all these contexts, a central problem of interest is to approximate the marginals of the path distribution (71). Above we already established the equivalence of DP programming and Newton steps in an LQ setting. Message passing in Gaussian factor graphs is generally equivalent to DP with squared cost-to-go functions. Typically one distinguishes between DP, which computes cost-to-go functions, and the forward unrolling of the optimal controller, to compute the optimal path. This can be viewed more symmetrically: computing optimal cost-to-go and cost-so-far functions forward and backward (or cost-to-go functions for all branches of a tree) equally yields the optimal path.

If the factor graph is not Gaussian, or the objective not 2nd-order polynomial, message passing as well as DP are approximated. Again, using Gaussian approximate message passing—e.g., as in extended Kalman filtering and smoothing—is equivalent to approximating the cost-to-go function locally as quadratic (Toussaint, 2009a). In conclusion, iterative Gaussian message passing to estimate marginals of (71) is very closely related to iterative DP using local LQG approximations and Newton methods to minimize (68).

So what are the motivations for the mentioned family of methods that build on the probabilistic interpretation? Essentially it is specific ideas on how exactly to do the approximation that arise from the probabilistic view, other than the Laplace approximation. For instance, in the probabilistic setting Gaussian messages can also be approximated using the Unscented Transform (Julier and Uhlmann, 1997), or Expectation Propagation (Minka, 2001). These are slightly different to local Laplace approximations. Importantly, if the path

[11] Why is this a natural relation? Let us assume we have $p(x)$. We want to find a *cost* quantity $f(x)$ which is some function of $p(x)$. We require that if a certain value x_1 is more likely than another, $p(x_1) > p(x_2)$, then picking x_1 should imply less cost, $f(x_1) < f(x_2)$ (Axiom 1). Further, when we have two independent random variables x and y *probabilities are multiplicative*, $p(x, y) = p(x)p(y)$. We require that, for independent variables, *cost is additive*, $f(x, y) = f(x) + f(y)$ (Axiom 2). From both follows that f needs to be a logarithm of p.

distribution cannot well be approximated as Gaussian, esp. because it is multi-modal, the probabilistic view provides alternative approaches to approximation, for instance, sampling from the path distribution (Kalakrishnan et al., 2011). Here we see that the goal of optimal control under a multi-modal path distribution really deviates from just computing an optimal path.

Incorporating hard constraints in approximate message passing is hard. In the context of Gaussian messages, truncated Gaussians could be used to approximate hard constraints (Toussaint, 2009b). However, in our experience this is far less precise and robust than using Lagrangian methods in optimization. Arguably, the handling of constraints, as well as the availability of robust optimization methods are the most important arguments in favor of the optimization view in comparison to the probabilistic interpretations. Multi-modality and true stochastic optimal control under such multi-modality are the strongest arguments for the probabilistic view.

As a side node on parallelizing message passing computations: KOMO, DDP, and iLQG all do full path updates in each iteration, that is, they compute a full new path $x_{0:T}$ in each Newton(-like) or line search step. This is a difference to AICO which allows to update individual states x_t in arbitrary order, not necessarily sweeping forward and backward. E.g. in AICO we can update a single x_t multiple times in sequence when the updates are large and therefore the local linearization changes significantly locally. This is possible because AICO computes backward *and* forward messages which define a local posterior belief for x_t that includes forward and backward messages. In the dynamic programming view this means that cost-to-go *and* cost-so-far functions are computed to define a local optimization problem over x_t only. In practice, however, these local path updates are harder to handle than global steps, especially because global monotonicity, as guaranteed by global Wolfe conditions, does not easily realized.

5 Conclusion

In this tutorial we chose the k-order cost and constraint feature convention to represent trajectory optimization problems as NLP. The implied structure of the Jacobians and Hessian is of central importance to understand the complexity of Newton steps in such settings.

Newton approaches are not just one alternative under many—they are at the core of efficient optimization as well as at the core of understanding the fundaments of the many related approaches mentioned in this tutorial. In particular, we've discussed the structure and complexity of computing Newton steps for banded symmetric Hessians and its relation to solving (tree- or Markov-) structured least squares problems using dynamic programming, both of which have a computational complexity linear in the number of variables. We have discussed control in the KOMO convention, especially constrained k-order dynamic programming to compute an approximate regulator around the optimal path with guaranteed feasibility, and its MPC extension. For the unconstrained LQ case we high-lighted the relations to DDP, iLQG, and AICO.

An interesting line of future research based on this discussion is related to path optimization processes that are not strictly Markovian in the KOMO sense. One example is jointly optimizing over paths and model parameters, which equally implies non-banded terms in the Hessian (Kolev and Todorov, 2015). Another example is sequential manipulation, in which costs that arise in some later part of the motion may directly depend on configuration decisions (grasps) made much earlier. The gradient of such costs then will always be non-zero w.r.t. the grasp configuration. These introduce "loops" in the dependencies that violate the k-order Markov assumption. However, Graph-SLAM has successfully addressed exactly this problem. The established relations between path optimization and Graph-SLAM may therefore be a promising candidate for an optimization-based approach to sequential manipulation.

Acknowledgement This work was supported by the DFG under grants TO 409/9-1 and the 3rdHand EU-Project FP7-ICT-2013-10610878.

References

R. Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences*, 42(10):767–769, 1956.

A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

J. T. Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.

A. R. Conn, N. I. Gould, and P. Toint. A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572, 1991.

F. Dellaert. Factor graphs and GTSAM: A hands-on introduction. Technical Report Technical Report GT-RIM-CP&R-2012-002, Georgia Tech, 2012.

M. Diehl, H. J. Ferreau, and N. Haverbeke. Efficient numerical methods for nonlinear MPC and moving horizon estimation. In *Nonlinear Model Predictive Control*, pages 391–417. Springer, 2009.

C. R. Dohrmann and R. D. Robinett. Dynamic programming method for constrained discrete-time optimal control. *Journal of Optimization Theory and Applications*, 101(2):259–283, 1999.

J. Dong, M. Mukadam, F. Dellaert, and B. Boots. Motion Planning as Probabilistic Inference using Gaussian Processes and Factor Graphs. In *Proceedings of Robotics: Science and Systems (RSS-2016)*, 2016.

P. Englert and M. Toussaint. Inverse KKT–Learning Cost Functions of Manipulation Tasks from Demonstrations. In *Proceedings of the International Symposium of Robotics Research*, 2015.

J. Folkesson and H. Christensen. Graphical SLAM-a self-correcting map. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 383–390. IEEE, 2004.

G. H. Golub and C. F. Van Loan. *Matrix Computations*, volume 3. JHU Press, 2012.

S. J. Julier and J. K. Uhlmann. New extension of the Kalman filter to nonlinear systems. In *AeroSense'97*, pages 182–193. International Society for Optics and Photonics, 1997.

M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4569–4574. IEEE, 2011.

H. J. Kappen, V. Gómez, and M. Opper. Optimal control as a graphical model inference problem. *Machine learning*, 87(2):159–182, 2012.

S. Kolev and E. Todorov. Physically consistent state estimation and system identification for contacts. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 1036–1043. IEEE, 2015.

F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519, 2001.

R. Kümmerle, 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*, pages 3607–3613. IEEE, 2011.

J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning (ICML)*, pages 282–289, 2001.

L.-z. Liao and C. A. Shoemaker. Advantages of differential dynamic programming over Newton's method for discrete-time optimal control problems. Technical report, Cornell University, 1992.

D. Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1):85–95, 1966.

T. P. Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

J. Nocedal and S. Wright. *Numerical Optimization*. Springer Science & Business Media, 2006.

J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008.

N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009.

N. Ratliff, M. Toussaint, and S. Schaal. Understanding the geometry of workspace obstacles in Motion Optimization. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4202–4209. IEEE, 2015.

K. Rawlik, M. Toussaint, and S. Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *Proc. of Robotics: Science and Systems (R:SS 2012)*, 2012. *Runner Up Best Paper Award*.

J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel. Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization. In *Robotics: Science and Systems*, volume 9, pages 1–10. Citeseer, 2013.

Y. Tassa, N. Mansard, and E. Todorov. Control-limited differential dynamic programming. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1168–1175. IEEE, 2014.

S. Thrun and M. Montemerlo. The graph SLAM algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006.

E. Todorov and W. Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 300–306. IEEE, 2005.

M. Toussaint. Robot trajectory optimization using approximate inference. In *Proc. of the Int. Conf. on Machine Learning (ICML 2009)*, pages 1049–1056. ACM, 2009a. ISBN 978-1-60558-516-1.

M. Toussaint. Pros and cons of truncated Gaussian EP in the context of Approximate Inference Control. NIPS Workshop on Probabilistic Approaches for Robotics and Control, 2009b.

M. Toussaint. A novel augmented lagrangian approach for inequalities and convergent any-time non-central updates. e-Print arXiv:1412.4329, 2014a.

M. Toussaint. KOMO: Newton methods for k-order markov constrained motion problems. e-Print arXiv:1407.0414, 2014b.

N. Vlassis and M. Toussaint. Model-free reinforcement learning as mixture learning. In *Proc. of the Int. Conf. on Machine Learning (ICML 2009)*, pages 1081–1088, 2009. ISBN 978-1-60558-516-1.

O. Von Stryk and R. Bulirsch. Direct and indirect methods for trajectory optimization. *Annals of operations research*, 37(1):357–373, 1992.