Temporal Parallelization of Dynamic Programming and Linear Quadratic Control

Simo Särkkä, *Senior Member, IEEE*, and Ángel F. García-Fernández

***Abstract*—This article proposes a general formulation for temporal parallelization of dynamic programming for optimal control problems. We derive the elements and associative operators to be able to use parallel scans to solve these problems with logarithmic time complexity rather than linear time complexity. We apply this methodology to problems with finite state and control spaces, linear quadratic tracking control problems, and to a class of nonlinear control problems. The computational benefits of the parallel methods are demonstrated via numerical simulations run on a graphics processing unit.**

***Index Terms*—Associative operator, dynamic programming, graphics processing unit (GPU), multicore processing, optimal control, parallel computing.**

I. Introduction

OPTIMAL control theory (see, e.g., [1]–[3]) is concerned with designing control signals to steer a system such that a given cost function is minimized, or equivalently, a performance measure is maximized. The system can be, for example, an airplane or autonomous vehicle, which is steered to follow a given trajectory, an inventory system, a chemical reaction, or a mobile robot [1], [4]–[7].

Dynamic programming, in the form first introduced by Bellman in the 1950s, is a general method for determining feedback laws for optimal control and other sequential decision problems [3], [8]–[10], and it also forms the basis of reinforcement learning [7], which is a subfield of machine learning. The classic dynamic programming algorithm is a sequential procedure that proceeds backward from the final time step to the initial time step, and determines the value (cost-to-go) function as well as the optimal control law in time complexity of $O(T)$, where T is the number of time steps. The algorithm is optimal in the sense that no sequential algorithm that processes all the T time steps can have a time complexity less than $O(T)$.

However, the complexity $O(T)$ is only optimal in a computer with one single-core central processing unit (CPU). Nowadays, even general-purpose computers typically have multicore CPUs with tens of cores and higher end computers can have hundreds of them. Furthermore, graphics processing units (GPUs) have become common accessories of general-purpose computers and current high-end GPUs can have tens of thousands of computational cores that can be used to parallelize computations and lower the time complexity.

Dynamic programming algorithms that parallelize computations at each time step, but operate sequentially, are provided in [11] and [12] for discrete states, and in [13] for the Riccati recursion in linear quadratic problems. Another approach to speed up computations for model predictive control (MPC) in linear quadratic problems is partial condensing [14], [15], which is based on splitting the problem into temporal blocks and eliminating the intermediate states algebraically. The required block conversion can be done in parallel and the resulting modified linear quadratic problem can be solved using parallel methods such as in [13]. However, the complexity of the resulting algorithm is still linear in time.

The previous dynamic programming algorithms have linear time-complexity $O(T)$, but there are some approaches in literature to lower this complexity by using parallelization across time. One idea applied in the context of an allocation process can be found in [9, Sec. I.30], where the time interval is divided into two, and the two problems are solved in parallel. Various forms of parallel algorithms for dynamic programming with discrete states are given in [16]. Wright [17] presented a partitioned dynamic programming suitable for parallelization for linear quadratic control problems, though it has the disadvantage that some required inverse matrices may not exist. An iterated method for linear quadratic control problems, with constraints, in which each step can be parallelized is proposed in [18], though it requires positive definite matrices in the cost function and may require regularization. An algorithm to approximately solve an optimal control problem by solving different subproblems with partially overlapping time windows is provided in [19], and an approximate parallel algorithm for linear MPC is given in [20]. Calvet *et al.* [21] provided combination rules to separate the dynamic programming algorithm into different subproblems across the temporal domain. These combination rules are the foundation for temporal parallelization.

The main contribution of this article is to present a parallel formulation of dynamic programming that is exact and has a time

Manuscript received 8 November 2021; accepted 16 January 2022. Date of publication 31 January 2022; date of current version 30 January 2023. The work of Simo Särkkä was supported by Academy of Finland. Recommended by Associate Editor S. S. Saab. *(Corresponding author: Simo Särkkä.)*

Simo Särkkä is with the Department of Electrical Engineering and Automation, Aalto University, 02150 Espoo, Finland (e-mail: simo.sarkka@aalto.fi).

Ángel F. García-Fernández is with the Department of Electrical Engineering and Electronics, University of Liverpool, L69 3GJ Liverpool, U.K., and also with the ARIES Research Centre, Universidad Antonio de Nebrija, 28015 Madrid, Spain (e-mail: angel.f.garcia.fernandez@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TAC.2022.3147017.

Digital Object Identifier 10.1109/TAC.2022.3147017

complexity $O(\log T)$. None of the previous works achieve these two aspects simultaneously. The central idea is to reformulate dynamic programming in terms of associative operators, which enable the use of parallel scan algorithms [22], [23] to parallelize the algorithm. The resulting algorithm has a span complexity of $O(\log T)$, which translates into a time complexity of $O(\log T)$ with a large enough number of computational cores. The algorithm can, therefore, speed up the computations significantly for long time horizons.

In this article, we first provide the general formulation to parallelize dynamic programming by defining conditional value functions between two different time steps and combining them via the rule in [21]. We also show how to obtain the optimal control laws and resulting trajectories making use of parallel computation. Then, we explain how this general methodology can be directly applied to problems with finite state and control spaces. The second contribution of this article is to specialize the methodology to linear quadratic optimal control problems, that is, to linear quadratic trackers (LQTs). The parallel LQT formulation is not straightforward, as it requires the propagation of the dual function [24] associated with the conditional value function to avoid numerical problems. Our third contribution is to extend the parallel LQT algorithm to approximately solve certain nonlinear control problems by iterated linearizations, as in [25]. Finally, we have implemented these algorithms in TensorFlow [26], which enables parallel computations on GPUs, to experimentally show that the parallel algorithms provide a significant speed up also in practice.

The present approach is closely related to the temporal parallelization of Bayesian smoothers and hidden Markov model inference recently considered in [27]–[29]. These approaches use a similar scan-algorithm-based parallelization in the context of state-estimation problems. The combination rule is also related to so-called max-plus algebras for dynamic programming, which have been considered, for example, in [30]–[33].

The structure of the article is the following. In Section II we provide a brief background on dynamic programming and parallel computing, in Section III we provide the parallel methods to general and finite-state problems, in Section IV we consider the parallel solution to LQT problems, in Section V, we discuss some practical implementation aspects and computational complexity, in Section VI we experimentally illustrate the performance of the methods on a GPU platform, and finally Section VII concludes this article.

II. Background

We provide a brief background on optimal deterministic control and its dynamic programming solution in Section II-A, the LQT case in Section II-B, and the parallel scan algorithm in Section II-C.

A. Deterministic Control Problem

We consider a deterministic control problem that consists of a difference equation and a cost function of the form [34]

$$\begin{aligned} x_{k+1} &= f_k(x_k, u_k) \\ C[u_{S:T-1}] &= \ell_T(x_T) + \sum_{n=S}^{T-1} \ell_n(x_n, u_n) \end{aligned} \tag{1}$$

where, for $k = S, \ldots, T$, x_k is the state (typically $x_k \in \mathbb{R}^{n_x}$); $f_k(\cdot)$ is the function that models the state dynamics at time step k; $u_{S:T-1} = (u_S, \ldots, u_{T-1})$ is the control/decision sequence (typically $u_k \in \mathbb{R}^{n_u}$ with $n_u \leq n_x$); and $\ell_k(\cdot)$ is a lower bounded function that indicates the cost at time step k. The initial state x_S is known. The aim is now to find a feedback control law or policy $u_k(x_k)$, such that if at step k the state is x_k, the cost function $C[u_{S:T-1}]$ for the steps from S to T is minimized with the sequence $u_S(x_S), \ldots, u_T(x_T)$.

In Bellman's dynamic programming [3], [8], [9], the idea is to form a cost-to-go or value function $V_k(x_k)$, which gives the cost of the trajectory when we follow the optimal decisions for the remaining steps up to T starting from state x_k. It can be shown [8] that the value function admits the recursion

$$V_k(x_k) = \min_{u_k} \{\ell_k(x_k, u_k) + V_{k+1}(f_k(x_k, u_k))\} \tag{2}$$

with $V_T(x_T) = \ell_T(x_T)$, which determines the optimal control law via

$$u_k(x_k) = \arg\min_{u_k} \{\ell_k(x_k, u_k) + V_{k+1}(f_k(x_k, u_k))\}. \tag{3}$$

Given the control law (3) for all time steps, we can compute the optimal trajectory from time steps $S+1$ to T, which is denoted as $(x^*_{S+1}, \ldots, x^*_T)$, by an additional forward pass starting at $x^*_S = x_S$ and

$$x^*_{k+1} = f_k(x^*_k, u_k(x^*_k)) = f^*_k(x^*_k). \tag{4}$$

B. Linear Quadratic Tracker

The LQT problem [2] is the solution to a linear quadratic control problem of the form

$$\begin{aligned} x_{k+1} &= F_k x_k + c_k + L_k u_k \\ \ell_T(x_T) &= \frac{1}{2}(H_T x_T - r_T)^\top X_T (H_T x_T - r_T) \\ \ell_n(x_n, u_n) &= \frac{1}{2}(H_n x_n - r_n)^\top X_n (H_n x_n - r_n) + \frac{1}{2} u_n^\top U_n u_n \end{aligned} \tag{5}$$

for $n = S, \ldots, T-1$. We assume that X_n and U_n are symmetric matrices, such that $X_n \geq 0$, $U_n > 0$.

In this setting, the objective is that a linear combination of the states $H_k x_k$ follows a reference trajectory r_k from time step S to T. The linear quadratic regulator is a special case of the LQT problem by setting $r_k = 0$, $c_k = 0$, and $H_k = I \; \forall k$.

In this case, the value function is

$$V_k(x_k) = \mathrm{z} + \frac{1}{2} x_k^\top S_k x_k - v_k^\top x_k \tag{6}$$

where v_k is an $n_x \times 1$ vector and S_k is an $n_x \times n_x$ symmetric matrix and, throughout the article, we use z to denote an undetermined constant that does not affect the calculations. The parameters v_k and S_k can be obtained recursively backward. Starting with $v_T = H_T^\top X_T r_N$ and $S_T = H_T^\top X_T H_T$, we obtain

$$v_k = (F_k - L_k K_k)^\top (v_{k+1} - S_{k+1} c_k) + H_k^\top X_k r_k \tag{7}$$

$$S_k = F_k^\top S_{k+1} (F_k - L_k K_k) + H_k^\top X_k H_k \tag{8}$$

where

$$K_k = \left(L_k^\top S_{k+1} L_k + U_k\right)^{-1} L_k^\top S_{k+1} F_k \tag{9}$$

and $k = S, \ldots, T-1$.

The optimal control law is

$$u_k = -K_k x_k + K_k^v v_{k+1} - K_k^c c_k \tag{10}$$

where

$$K_k^v = \left(L_k^\top S_{k+1} L_k + U_k\right)^{-1} L_k^\top \tag{11}$$

$$K_k^c = \left(L_k^\top S_{k+1} L_k + U_k\right)^{-1} L_k^\top S_{k+1}. \tag{12}$$

The optimal trajectory resulting from applying the optimal control law (10) can be computed with an additional forward pass (4) starting at x_S and with control law (10).

It should be noted that the derivation of LQT in [2] does not include time-varying matrices or parameter c_k in the problem formulation (5), but it is straightforward to include these. It is also possible to use the LQT solution as a basis for approximate nonlinear control by linearizing the system along a nominal trajectory (see [1], [25], [35], and Section IV-D3).

C. Associative Operators and Parallel Computing

Parallel computing (see, e.g., [36] and [37]) refers to programming and algorithm design methods that take the availability of multiple computational cores into account. When some parts of the problem can be solved independently, then those parts can be solved in parallel to reduce the computational time. The more parts we can solve in parallel, the more speed up we get.

Sequential problems, which at first glance do not seem to be parallelizable, can often be parallelized using the so called parallel scan or all-prefix-sums algorithms [22], [23]. Given a sequence of elements $a_1, \ldots, a_T$ and an associative operator $\otimes$ defined on them, such as summation, multiplication, or minimization, the parallel scan algorithm computes the all-prefix-sums operation which returns the values $s_1, \ldots, s_T$ such that

$$\begin{aligned} s_1 &= a_1 \\ s_2 &= a_1 \otimes a_2 \\ &\cdots \\ s_T &= a_1 \otimes a_3 \otimes \cdots \otimes a_T \end{aligned} \tag{13}$$

in $O(\log T)$ time. The key aspect is that, because the operator $\otimes$ is associative, we can rearrange the computations in various ways which generate independent subproblems, for example,

$$((a_1 \otimes a_2) \otimes a_3) \otimes a_4 = (a_1 \otimes a_2) \otimes (a_3 \otimes a_4) \tag{14}$$

and, by a suitable combination of the partial solutions, we can obtain the result in $O(\log T)$ parallel steps. The specific combination requires an up-sweep and a down-sweep on a binary tree of computations [23]. A pseudocode is given in Algorithm 1. Clearly, the prefix sums can also be computed in parallel in the backward direction $(a_1 \otimes \cdots \otimes a_T, \ldots, a_{T-1} \otimes a_T, a_T)$.

It should be noted that, while parallel scans significantly lower the wall-clock time to compute all-prefix-sums, they have the drawback that the number of total computations is higher than in the sequential algorithm [23]. This implies that they require higher energy, which may not be suitable for small-scale mobile systems.

Algorithm 1: Parallel-Scan Algorithm. The algorithm in this form assumes that T is a power of 2, but it can easily be generalized to an arbitrary T.

```
Input: The elements $a_k$ for $k = 1, \ldots, T$ and an
  associative operator $\otimes$.
Output: The all prefix sums are returned in $a_k$ for
  $k = 1, \ldots, T$.
1:  // Save the input:
2:  for $i \leftarrow 1$ to $T$ do {Compute in parallel}
3:    $b_i \leftarrow a_i$
4:  end for
5:  // Up-sweep:
6:  for $d \leftarrow 0$ to $\log_2 T - 1$ do
7:    for $i \leftarrow 0$ to $T - 1$ by $2^{d+1}$ do {Compute in
      parallel}
8:      $j \leftarrow i + 2^d$
9:      $k \leftarrow i + 2^{d+1}$
10:     $a_k \leftarrow a_j \otimes a_k$
11:   end for
12: end for
13: $a_T \leftarrow 0$ {Here, 0 is the neutral element for $\otimes$}
14: // Down-sweep:
15: for $d \leftarrow \log_2 T - 1$ to 0 do
16:   for $i \leftarrow 0$ to $T - 1$ by $2^{d+1}$ do {Compute in
      parallel}
17:     $j \leftarrow i + 2^d$
18:     $k \leftarrow i + 2^{d+1}$
19:     $t \leftarrow a_j$
20:     $a_j \leftarrow a_k$
21:     $a_k \leftarrow a_k \otimes t$
22:   end for
23: end for
24: // Final pass:
25: for $i \leftarrow 1$ to $T$ do {Compute in parallel}
26:   $a_i \leftarrow a_i \otimes b_i$
27: end for
```

III. Parallel Optimal Control

In this section, we start by defining conditional value functions and their combination rules (Section III-A) and then we use them to define the associative operators and elements for parallelization (Section III-B). We also derive the parallel solution of the resulting optimal trajectory (Section III-C), and finally, we discuss the case where the state space and controls take values in finite sets (Section III-D).

A. Conditional Value Functions and Combination Rules

In this section, we present the conditional value functions and their combination rules, which are required to design the parallel algorithms.

Definition 1 (Conditional value function): The conditional value function $V_{k\to i}(x_k, x_i)$ is the cost of the optimal trajectory starting from x_k and ending at x_i, that is,

$$V_{k\to i}(x_k, x_i) = \min_{u_{k:i-1}} \sum_{n=k}^{i-1} \ell_n(x_n, u_n) \tag{15}$$

subject to

$$x_n = f_{n-1}(x_{n-1}, u_{n-1}) \quad \forall n \in \{k+1, \ldots, i\}. \tag{16}$$

If there is no path connecting x_k and x_i, then the constraint (16) cannot be met and $V_{k\to i}(x_k, x_i) = \infty$.

The combination rule for conditional value functions is provided in the following theorem.

Theorem 2: The recursions for the value functions and conditional value functions can be written as

$$V_{k\to i}(x_k, x_i) = \min_{x_j} \{V_{k\to j}(x_k, x_j) + V_{j\to i}(x_j, x_i)\} \quad (17)$$

for $k < j < i \leq T$ and

$$V_k(x_k) = \min_{x_i} \{V_{k\to i}(x_k, x_i) + V_i(x_i)\} \quad (18)$$

for $k < i \leq T$.

Proof: See Appendix A. ■

As part of the minimization in (18), we also get the minimizing state x_i. Due to the principle of optimality, this value is the state at time step i, that is, on the optimal trajectory from x_k until time T. Similarly, the argument of minimization x_j in (17) is part of the optimal trajectory from x_k to x_i.

B. Associative Operator for Value Functions

The associative element a of the parallel scan algorithm is defined to be a conditional value function $V_a(\cdot,\cdot) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to \mathbb{R}$ such that

$$a = V_a(x, y). \quad (19)$$

The combination rule for two elements $a = V_a(x, y)$ and $b = V_b(x, y)$ is then given as follows.

Definition 3: Given elements a and b of the form (19), the binary associative operator for dynamic programming is

$$a \otimes b \triangleq \min_z \{V_a(x, z) + V_b(z, y)\}. \quad (20)$$

This operator is an associative operator because min operation is associative. This is summarized in the following lemma.

Lemma 4: The operator in Definition 3 is associative.

Proof: For three elements a, b, and c, we have

$$\begin{aligned} &(a \otimes b) \otimes c \\ &\triangleq \min_{z'} \left\{ \min_z \{V_a(x, z) + V_b(z, z')\} + V_c(z,' y) \right\} \\ &= \min_z \left\{ V_a(x, z) + \min_{z'} \{V_b(z, z') + V_c(z,' y)\} \right\} \\ &\triangleq a \otimes (b \otimes c) \end{aligned} \quad (21)$$

which shows that $(a \otimes b) \otimes c = a \otimes (b \otimes c)$. ■

The elements and combination rule allows us to construct the conditional and conventional value functions as follows.

Theorem 5: If we initialize the elements a_k for $k = S, \ldots, T$ as

$$a_k = V_{k\to k+1}(x_k, x_{k+1}) \quad (22)$$

where $V_{T\to T+1}(x_T, x_{T+1}) \triangleq V_T(x_T)$, then

$$a_S \otimes a_{S+1} \otimes \cdots \otimes a_{k-1} = V_{S\to k}(x_S, x_k) \quad (23)$$

and

$$a_k \otimes a_{i+1} \otimes \cdots \otimes a_T = V_k(x_k). \quad (24)$$

Proof: Equation (23) results from the sequential application of (17) forward, and (24) from the sequential application (18) backward. ■

Theorem 5 implies that we can compute all value functions $V_k(\cdot)$ by initializing the elements as in (22), using the associative operator in Definition 3 and computing (24) for $k = S, \ldots, T - 1$, which corresponds to a (reverted) all-prefix-sum operation. Because the initialization is fully parallelizable, we can directly use the parallel scan algorithm (see Algorithm 1) to compute all value functions in $O(\log T)$ parallel steps.

Remark 6: After computing all the value functions, we can obtain all the control laws $u_k(x_k)$ for $k = S, \ldots, T - 1$ by using (3). This operation can be done in parallel for each k.

Remark 7: If we are interested in evaluating $V_{S\to k}(x_S, x_k)$ at a given x_S, as we are in trajectory recovery, then instead of first using (23) with initialization (22), and then evaluating the result at x_S, we can also initialize an extra element

$$a_{S-1} = V_{S-1\to S}(x, x') = \begin{cases} 0 & \text{if } x' = x_S \\ \infty & \text{otherwise.} \end{cases} \quad (25)$$

C. Optimal Trajectory Recovery

Once we have obtained the optimal control laws (3) in parallel, we can compute the resulting optimal trajectory $(x^*_{S+1}, \ldots, x^*_T)$ in parallel using two methods.

1) Method 1: In the first method for trajectory recovery, the state of the optimal trajectory at time step k can be computed by using (4) and the composition of functions

$$x^*_k = \left(f^*_{k-1} \circ \ldots \circ f^*_{S+1} \circ f^*_S\right)(x_S). \quad (26)$$

We can compute (26) using parallel scans as follows. The associative element a is defined to be a function on x, $a = f_a(\cdot)$ and the operator is the function composition in the following definition.

Definition 8: Given elements $a = f_a(\cdot)$ and $b = f_b(\cdot)$, the binary associative operator for optimal trajectory recovery is

$$a \otimes b \triangleq f_b \circ f_a \quad (27)$$

where $\circ$ denotes the composition of two functions, which is an associative operator [38]. We should note that the order of the function composition is reverted.

Then, we can recover the optimal trajectory via the following lemma.

Lemma 9: If we initialize element a_S as the function $f^*_S(\cdot)$, which is given by (4), evaluated at x_S

$$a_S = f^*_S(x_S) \quad (28)$$

and, for $k = S + 1, \ldots, T - 1$, a_k is initialized as the function

$$a_k = f^*_k(\cdot) \quad (29)$$

then

$$a_S \otimes a_{S+1} \otimes \cdots \otimes a_{k-1} = x^*_k \quad (30)$$

where x^*_k is the state of the optimal trajectory at time step k.

2) Method 2: An alternative method, which resembles the max-product algorithm in probabilistic graphical models [39], is based on noticing that, from the definition of the conditional value function (15) and the value function (2), the state of the optimal trajectory at time step k is given by

$$x^*_k = \arg\min_{x_k} \{V_{S\to k}(x_S, x_k) + V_k(x_k)\} \quad (31)$$

where we recall that x_S is the initial known state. That is, we can just minimize the sum of the forward conditional value function $V_{S\to k}(x_S, x_k)$ and the (backward) value function $V_k(x_k)$, which can be calculated using parallel scans via (23) and (24), respectively. Then, the minimization (31) can be done for each node in parallel.

It should be noted that both approaches for optimal trajectory recovery require two parallel scans, one forward and one backwards, and one minimization for each node.

D. Finite State and Control Spaces

The case in which the state and the control input belong to finite state spaces is important as we can solve the control problem in both sequential and parallel forms in closed form. Let $x_k \in \{1, \ldots, D_x\}$ and $u_k \in \{1, \ldots, D_u\}$, where D_x and D_u are natural numbers. Then, $f_k(x_k, u_k)$ and $\ell_n(x_n, u_n)$ can be represented by matrices of dimensions $D_x \times D_u$, $V_k(x_k)$ by a vector of dimension D_x, $u_k(x_k)$ by a vector of dimension D_x, and $V_{k\to i}(x_k, x_i)$ by a matrix of size $D_x \times D_x$. Due to the finite state space, the required minimizations in (2) and (20), can be performed by exhaustive search, which can also be parallelized.

For Method 1 for optimal trajectory recovery, the function $f_k^*(\cdot)$ can be represented as a vector of dimension D_x, and the function composition in (31) can be performed by evaluating all cases.

IV. Parallel Linear Quadratic Tracker

In this section, we provide the parallel solution to the LQT case. In Section IV-A, we derive the conditional value functions and combination rules. In Section IV-B, we derive the associative elements to obtain the value functions $V_{S\to k}(x_S, x_k)$ and $V_k(x_k)$. In Section IV-C, we address the computation of the optimal trajectory. Finally, in Section IV-D, we discuss extensions to stochastic and nonlinear problems.

A. Conditional Value Functions and Combination Rules

For the LQT problem in (5), $V_{k\to i}(x_k, x_i)$ in (15) is a quadratic program with affine equality constraints [24] that we represent by its dual problem

$$V_{k\to i}(x_k, x_i) = \max_{\lambda} g_{k\to i}(\lambda; x_k, x_i) \tag{32}$$

where λ is a Lagrange multiplier $n_x \times 1$ vector and the dual function $g_{k\to i}(\cdot,\cdot,\cdot)$ has the parameterization

$$\begin{aligned} g_{k\to i}(\lambda; x_k, x_i) = \mathrm{z} &+ \frac{1}{2} x_k^\top J_{k,i} x_k - x_k^\top \eta_{k,i} \\ &- \frac{1}{2}\lambda^\top C_{k,i}\lambda - \lambda^\top (x_i - A_{k,i}x_k - b_{k,i}) . \end{aligned} \tag{33}$$

If $C_{k,i}$ is invertible, one can solve (32) by calculating the gradient of (33) with respect to λ and setting it equal to zero, to obtain

$$\begin{aligned} V_{k\to i}(x_k, x_i) = \mathrm{z} &+ \frac{1}{2} x_k^\top J_{k,i} x_k - x_k^\top \eta_{k,i} \\ &+ \frac{1}{2}(x_i - A_{k,i}x_k - b_{k,i})^\top \\ &\times C_{k,i}^{-1}(x_i - A_{k,i}x_k - b_{k,i}) . \end{aligned} \tag{34}$$

In this case, we can also interpret the conditional value function (34) in terms of conditional Gaussian distributions as

$$\begin{aligned} \exp(-V_{k\to i}(x_k, x_i)) &\propto \mathrm{N}(x_i; A_{k,i}x_k + b_{k,i}, C_{k,i}) \\ &\times \mathrm{N}_I(x_k; \eta_{k,i}, J_{k,i}) \end{aligned} \tag{35}$$

where $\mathrm{N}(\cdot; \overline{x}, P)$ denotes a Gaussian density with mean $\overline{x}$ and covariance matrix P, and $\mathrm{N}_I(\cdot; \eta, J)$ denotes a Gaussian density parameterized in information form with information vector η and information matrix J. A Gaussian distribution with mean $\overline{x}$ and covariance matrix P can be written in its information form as $\eta = P^{-1}\overline{x}$ and $J = P^{-1}$.

Nevertheless, in general, $C_{k,i}$ is not invertible so it is suitable to keep the dual function parameterization in (32).

Lemma 10: Given two elements $V_{k\to j}(x_k, x_j)$ and $V_{j\to i}(x_j, x_i)$ of the form (32), their combination $V_{k\to i}(x_k, x_i)$, which is obtained using Theorem 2, is of the form (32) and characterized by

$$\begin{aligned} A_{k,i} &= A_{j,i}(I + C_{k,j}J_{j,i})^{-1}A_{k,j} \\ b_{k,i} &= A_{j,i}(I + C_{k,j}J_{j,i})^{-1}(b_{k,j} + C_{k,j}\eta_{j,i}) + b_{j,i} \\ C_{k,i} &= A_{j,i}(I + C_{k,j}J_{j,i})^{-1}C_{k,j}A_{j,i}^\top + C_{j,i} \\ \eta_{k,i} &= A_{k,j}^\top(I + J_{j,i}C_{k,j})^{-1}(\eta_{j,i} - J_{j,i}b_{k,j}) + \eta_{k,j} \\ J_{k,i} &= A_{k,j}^\top(I + J_{j,i}C_{k,j})^{-1}J_{j,i}A_{k,j} + J_{k,j} \end{aligned} \tag{36}$$

where I is an identity matrix of size n_x.

The proof is provided in Appendix A. It should be noted that the combination rule (36) is equivalent to the combination rule for the parallel linear and Gaussian filter, which also considers Gaussian densities of the form (35) [27, Lemma 8].

B. Associative Elements to Obtain the Value Functions

The following lemma establishes how to define the elements of the parallel scan algorithms to obtain the value functions $V_{S\to k}(x_S, x_k)$ and $V_k(x_k)$.

Lemma 11: If we initialize the elements a_k for $k = S, \ldots, T$ as

$$a_k = V_{k\to k+1}(x_k, x_{k+1}) \tag{37}$$

where $V_{k\to k+1}(x_k, x_{k+1})$ is of the form (32) with

$$\begin{aligned} A_{k,k+1} &= F_k \\ b_{k,k+1} &= c_k \\ C_{k,k+1} &= L_k U_k^{-1} L_k^\top \\ \eta_{k,k+1} &= H_k^\top X_k r_k \\ J_{k,k+1} &= H_k^\top X_k H_k \end{aligned} \tag{38}$$

for $k = S, \ldots, T-1$ and $V_{T\to T+1}(x_T, x_{T+1})$ has parameters

$$\begin{aligned} A_{T,T+1} &= 0 \\ b_{T,T+1} &= 0 \\ C_{T,T+1} &= 0 \\ \eta_{T,T+1} &= H_T^\top r_T \\ J_{T,T+1} &= H_T^\top X_T H_T \end{aligned} \tag{39}$$

then

$$a_S \otimes a_{S+1} \otimes \cdots \otimes a_{k-1} = V_{S\to k}(x_S, x_k) \tag{40}$$

and

$$a_k \otimes a_{i+1} \otimes \cdots \otimes a_T = V_{k\to T+1}(x_k, x_{T+1}) \tag{41}$$

where

$$V_{k\to T+1}(x_k, 0) = V_k(x_k). \tag{42}$$

Furthermore, $V_k(x_k)$ is of the form (6) with

$$\begin{aligned} S_k &= J_{k,T+1} \\ v_k &= \eta_{k,T+1}. \end{aligned} \tag{43}$$

Proof: This lemma is proved in Appendix A. ■

Once we obtain v_{k+1} and S_{k+1} using Lemma 11, we can compute the optimal control u_k using (10).

Remark 12: If we are interested in evaluating the conditional value functions $V_{S\to k}(x_S, x_k)$ for a given x_S, then we can also directly initialize by

$$\begin{aligned} A_{S-1,S} &= 0 \\ b_{S-1,S} &= x_S \\ C_{S-1,S} &= 0 \\ \eta_{S-1,S} &= 0 \\ J_{S-1,S} &= 0. \end{aligned} \tag{44}$$

C. Optimal Trajectory Recovery

We proceed to explain how the two optimal trajectory recovery methods explained in Section III-C work for the LQT problem.

1) Method 1: Plugging the optimal control law (10) into the dynamic equation in (5), the optimal trajectory function in (4) becomes

$$f_k^*(x_k) = \widetilde{F}_k x_k + \widetilde{c}_k \tag{45}$$

where

$$\widetilde{F}_k = F_k - L_k K_k \tag{46}$$

$$\widetilde{c}_k = c_k + L_k K_k^v v_{k+1} - L_k K_k^c c_k. \tag{47}$$

We denote a conditional optimal trajectory from time step k to i as

$$f_{k\to j}^*(x_k, x_j) = \left(f_{j-1}^* \circ \ldots \circ f_{k+1}^* \circ f_k^*\right)(x_k) \tag{48}$$

$$= \widetilde{F}_{k,j} x_k + \widetilde{c}_{k,j}. \tag{49}$$

Lemma 13: Given two elements $f_{k\to j}^*(x_k, x_j)$ and $f_{j\to i}^*(x_j, x_i)$ of the form (49), their combination $f_{k\to i}^*(x_k, x_i)$, given by Definition 8, is a function $f_{k\to i}^*(x_k, x_i)$ of the form (49) with

$$\widetilde{F}_{k,i} = \widetilde{F}_{j,i}\widetilde{F}_{k,j} \tag{50}$$

$$\widetilde{c}_{k,i} = \widetilde{F}_{j,i}\widetilde{c}_{k,j} + \widetilde{c}_{j,i}. \tag{51}$$

Proof: The proof of this lemma is direct by using function compositions. ■

How to recover the optimal trajectory using parallel scans is indicated in the following lemma.

Lemma 14: If we initialize the elements of the parallel scan as $a_k = f_{k\to k+1}^*(x_k)$, with

$$\widetilde{F}_{k,k+1} = \widetilde{F}_k \tag{52}$$

$$\widetilde{c}_{k,k+1} = \widetilde{c}_k \tag{53}$$

for $k \in \{S+1, \ldots, T-1\}$, and, for $k = S$, we set $\widetilde{F}_{S,S+1} = 0$ and $\widetilde{c}_{S,S+1} = \widetilde{F}_S x_S + \widetilde{c}_S$, then

$$a_S \otimes a_{S+1} \otimes \cdots \otimes a_{k-1} = x_k^* \tag{54}$$

where x_k^* is the state of the optimal trajectory at time step k.

2) Method 2: This method makes use of (31) to recover the optimal trajectory. It first runs a forward pass to compute $V_{S\to k}(x_S, x_k)$ and then a backward pass to compute $V_k(x_k)$. Then, the optimal trajectory is obtained via the following lemma.

Lemma 15: Given $V_k(x_k)$ of the form (6) and $V_{S\to k}(x_S, x_k)$ of the form (32), the state of the optimal trajectory at time step k, which is obtained using (31), is

$$x_k^* = (I + C_{S,k} S_k)^{-1} (A_{S,k} x_S + b_{S,k} + C_{S,k} v_k). \tag{55}$$

Proof: The proof is provided in Appendix A. ■

D. Extensions

In this section, the aim is to discuss some straightforward extensions of the parallel LQT.

1) Extension to Stochastic Control: Although the extension of the general framework introduced in this article to stochastic control problems is hard, the stochastic LQT case follows easily. Stochastic LQT is concerned with models of the form

$$x_{k+1} = F_k x_k + c_k + L_k u_k + G_k w_k \tag{56}$$

$$C[u_{S:T-1}] = \mathrm{E}\left[\ell_T(x_T) + \sum_{n=S}^{T-1} \ell_n(x_n, u_n)\right] \tag{57}$$

where $\ell_T(x_T)$ and $\ell_n(x_n)$ are as given in (5), and w_k is a zero mean white noise process with covariance Q_k, G_k is a given matrix, and $\mathrm{E}[\cdot]$ denotes expectation over the state trajectory. It turns out that due to certainty equivalence property of linear stochastic control problems [1], [35], the optimal control is still given by (10) and the solution exactly matches the deterministic solution, that is, it is independent of Q_k and G_k. The optimal value functions both in deterministic and stochastic cases have the form (6), but the value of the (irrelevant) constant is different.

It also results from the certainty equivalence property that the optimal control solution to the partially observed linear (affine) stochastic control problem with function (57) and dynamic and measurement models

$$x_{k+1} = F_k x_k + c_k + L_k u_k + G_k w_k \tag{58}$$

$$y_k = O_k x_k + d_k + e_k \tag{59}$$

where y_k is a measurement; O_k is a measurement model matrix; d_k is a deterministic bias; and e_k is a zero mean Gaussian measurement noise, given by (10), where the state x_k is replaced with its Kalman filter estimate.

2) Extension to More General Cost Functions: Sometimes (such as in the nonlinear case below) we are interested in generalizing the cost function in (5) to the following form for $n < T$:

$$\begin{aligned} \ell_n(x_n, u_n) &= \frac{1}{2}(H_n x_n - r_n)^\top X_n (H_n x_n - r_n) \\ &\quad + (H_n x_n - r_n)^\top M_n (u_n - s_n) \\ &\quad + \frac{1}{2}(u_n - s_n)^\top U_n (u_n - s_n) \\ &= \frac{1}{2}\begin{bmatrix} H_n x_n - r_n \\ u_n - s_n \end{bmatrix}^\top \begin{bmatrix} X_n & M_n \\ M_n^\top & U_n \end{bmatrix} \begin{bmatrix} H_n x_n - r_n \\ u_n - s_n \end{bmatrix}. \end{aligned} \tag{60}$$

We can now transform (60) into the form (5) using the factorization

$$\begin{pmatrix} X_n & M_n \\ M_n^\top & U_n \end{pmatrix} = \begin{pmatrix} I & 0 \\ U_n^{-1} M_n^\top & I \end{pmatrix}^\top \times \begin{pmatrix} X_n - M_n U_n^{-1} M_n^\top & 0 \\ 0 & U_n \end{pmatrix} \begin{pmatrix} I & 0 \\ U_n^{-1} M_n^\top & I \end{pmatrix}. \quad (61)$$

We thus have

$$\begin{pmatrix} I & 0 \\ U_n^{-1} M_n^\top & I \end{pmatrix} \begin{pmatrix} H_n x_n - r_n \\ u_n - s_n \end{pmatrix} = \begin{pmatrix} H_n x_n - r_n \\ U_n^{-1} M_n^\top (H_n x_n - r_n) + u_n - s_n \end{pmatrix} \quad (62)$$

and by defining

$$\begin{aligned} \tilde{u}_n &= U_n^{-1} M_n^\top (H_n x_n - r_n) + u_n - s_n \\ \tilde{F}_n &= F_n - L_n U_n^{-1} M_n^\top H_n \\ \tilde{c}_n &= c_n + L_n U_n^{-1} M_n^\top r_n + L_n s_n \\ \tilde{X}_n &= X_n - M_n U_n^{-1} M_n^\top \\ \tilde{U}_n &= U_n \end{aligned} \quad (63)$$

we get a system of the form (5). This system can then be solved for $(x_n, \tilde{u}_n)$, and the final control signal can be recovered via

$$u_n = \tilde{u}_n - U_n^{-1} M_n^\top (H_n x_n - r_n) + s_n. \quad (64)$$

3) Extension to Nonlinear Control: The equations for solving the LQT problem can be extended to approximately solve nonlinear LQT systems by performing iterated linearizations, as in [25]. Let us consider a system of the form

$$\begin{aligned} x_{k+1} &= f_k(x_k, u_k) \\ \ell_n(x_n, u_n) &= \frac{1}{2}(h_n(x_n) - r_n)^\top X_n (h_n(x_n) - r_n) \\ &\quad + \frac{1}{2}(g_n(u_n) - s_n)^\top U_n (g_n(u_n) - s_n) \\ \ell_T(x_T) &= \frac{1}{2}(h_T(x_T) - r_T)^\top X_T (h_T(x_T) - r_T) \end{aligned} \quad (65)$$

where $f_k(\cdot)$, $g_n(\cdot)$, and $h_n(\cdot)$ are possibly nonlinear functions. Given a nominal trajectory $\bar{x}_k, \bar{u}_k$ for $k \in S, \ldots, T$, we can linearize the nonlinear functions using first-order Taylor series as

$$\begin{aligned} f_k(x_k, u_k) &\approx f_k(\bar{x}_k, \bar{u}_k) + J^x_{f_k}(x_k - \bar{x}_k) + J^u_{f_k}(u_k - \bar{u}_k) \\ h_n(x_n) &\approx h_n(\bar{x}_n) + J^x_{h_n}(x_n - \bar{x}_n) \\ g_n(u_n) &\approx g_n(\bar{u}_n) + J^u_{g_n}(u_n - \bar{u}_n) \end{aligned} \quad (66)$$

where $J^x_{f_k}$ represents the Jacobian of function $f_k(\cdot)$ evaluated at $\bar{x}_k, \bar{u}_k$ with respect to variable x.

Starting with a nominal trajectory $\bar{x}^1_k, \bar{u}^1_k$ for $k \in S, \ldots, T$, we linearize the system using (66), obtain the value functions using parallel scans (see Lemma 11), and obtain a new optimal trajectory $\bar{x}^2_k$ and control $\bar{u}^2_k$. Then, we can repeat this procedure of linearization and optimal trajectory/control computation until convergence. The procedure may be initialized, for example, with $\bar{x}^1_k = 0, \bar{u}^1_k = 0$ or $\bar{x}^1_k = x_S, \bar{u}^1_k = 0 \; \forall k$.

V. Implementation and Computational Complexity

In this section, we first discuss the practical implementation of the methods in Section V-A. We then analyze the computational complexity in Section V-B. Finally, we explain how to perform parallelization in blocks in Section V-C.

A. Practical Implementation of Parallel Control

Given the associative operators and the elements, the solutions to the dynamic programming and trajectory prediction problems reduce to an initialization step followed by a single call to a parallel scan algorithm routine parameterized by these operators and elements. Given the result of the scan, there can also be a result-extraction step, which computes the final optimal control from the scan results. For example, the LQT control law computation consists of the following steps.

1) *Initialization:* Compute the elements $A_{k,k+1}$, $b_{k,k+1}$, $C_{k,k+1}$, $\eta_{k,k+1}$, and $J_{k,k+1}$ defined in Lemma 11 for all k in parallel.
2) *Parallel Scan:* Call the backward parallel scan routine and, as its arguments, give the initialized elements above along with pointer to the operator in Lemma 10. This returns $V_k(x_k)$ for all k, see (41) and (42).
3) *Extraction:* Compute the optimal control using (10) in parallel for all k.

The control law for a finite-state control problem is initialized with the conditional value functions in Theorem 5 and the parallel scan routine is given a pointer to the operator in Definition 3. The control law computation is finally done with (3) using the value functions computed in parallel.

Sometimes, we also need to compute the actual trajectory and the corresponding optimal controls forward in time. For example, in iterative nonlinear extensions of LQT discussed in Section IV-D we need to linearize the trajectory with respect to the optimal trajectory and control obtained at the previous iteration. In this case, after computing the control laws, we need to do another computational pass. For example, in the LQT case when using Method 1 from Section IV-C, we do the following.

1) *Initialization:* Compute the elements $\widetilde{F}_{k,k+1}$ and $\widetilde{c}_{k,k+1}$ using Lemma 14 for all k in parallel.
2) *Parallel Scan:* Call the forward parallel scan routine and, as its arguments, give the initialized elements above along with pointer to the operator given in Lemma 13.
3) *Extraction:* The optimal trajectory can be extracted from the forward scan results as the elements $\widetilde{c}_{S,k}$, see Lemma 14.

The steps for Method 1 in the finite-state case are analogous, but the elements are initialized according to Lemma 9 and the combination operator is given in Definition 8.

When using Method 2 for optimal trajectory recovery, the operator is the same as in the backward computation for the control law. The parallel scan is done in the forward direction and the final results still need to be evaluated at x_S unless initialization is done using Remark 12. Furthermore, after computing the backward and forward scans, we still need to compute the optimal states by using (31), which in the case of LQT reduces to (55)

B. Computational Complexity

We proceed to analyze the computational complexity of the proposed methods. For this purpose it is useful to assume that the computer that we have operates according to the parallel random access machine (PRAM) model of computation (see, e.g., [36] and [37]). In this model, we assume that we have a bounded-number P of identical processors controller by a common clock with a read/write access to a shared random access memory. This model is quite accurate for multicore CPUs and GPUs.

For simplicity of analysis, we assume that the number of processors is large enough (say $P \to \infty$), so that the number of processors does not limit the parallelization. Thus, the parallel scan algorithm has a time complexity (i.e, span-complexity) of $O(\log T)$ in the number of associative operations. In the following, we also take the dimensionality of the state into account and, therefore, the time complexities not only depend on the number of time steps T but also on the number of states D_x and number of controls D_u in finite state-space case, and dimensionalities of the state n_x and control n_u in the LQT case. We analyze complexity using $O(\cdot)$ notation, as it enables us to analyze computational complexity avoiding low-level operation details that are not relevant to this contribution [40].

The combination rule computations can also be parallelized, and their complexity will, therefore, depend on whether they are parallelized or not. For example, if we do not parallelize the computations in the LQT combination rule given in Lemma 10, then their complexity is $O(n_x^3)$ due to the matrix inverses (or equivalent LU-factorizations) involved. If we perform the parallelization, these LU-factorizations can be performed in parallel in $O(n_x)$ span time [41]. The following analysis is based on assuming that we in fact use parallel matrix routines to implement the operations at the combination steps, along with all the other steps.

For the parallel algorithms we obtain the following results.

Lemma 16: In a PRAM computer with large enough number of processors ($P \to \infty$) and the finite-state control problem, the span time complexity of

1) computing value functions and the control law is $O(\log D_u + (\log T)\,(\log D_x))$; and
2) recovering the trajectory is $O(\log T)$ with Method 1, and $O(\log D_u + \log D_x + (\log T)\,(\log D_x))$ with Method 2.

Proof: The initialization of the value function computation is done using (22) which has a time (span) complexity of $O(\log D_u)$ due to the minimization operation over the control input. The associative operator in Definition 3 is fully parallelizable in summation, but the span complexity of the minimization over the state is $O(\log D_x)$. The control law computation (3) also has the complexity $O(\log D_u)$ and hence the total span complexity follows. For trajectory recovery with Method 1, we notice that each of the steps of initialization and associative operator application are fully parallelizable. In Method 2, the initialization and value function computation have the same complexity as in the backward value function computation and the minimization at the final combination step takes $O(\log D_x)$ time. ■

Lemma 17: In a PRAM computer with large enough number of processors ($P \to \infty$) and the LQT problem, the span time complexity of

1) computing value functions and the control law is $O(n_u + n_x \log T)$; and
2) recovering the trajectory is $O(\log n_u + (\log T)\,(\log n_x))$ with Method 1, and $O(n_x + n_x \log T)$ with Method 2.

Proof: A product of $n \times n$ matrices can be computed in parallel in $O(\log n)$ time, and an $n \times n$ LU factorization can be computed in parallel in $O(n)$ time [41]. Hence the initialization requires $O(n_u)$ time as the contribution of the matrix products is negligible. The time complexity of the associative operator is dominated by the LU factorizations which take $O(n_x)$ time and the matrix factorizations at the control law computation can be performed in $O(n_u)$ time. The matrix products required in initialization have negligible effect and, therefore, the total complexity follows. In trajectory recovery Method 1, the matrix products at the initialization can be computed in $O(\log n_u)$ time and the associative operators in $O(\log n_x)$ time. In Method 2, the initialization is again negligible, and associative operations take $O(n_x)$ time, and the final combination $O(n_x)$ time. ■

It should be noted that, according to Lemmas 16 and 17, Method 1 is computationally more efficient than Method 2 for large D_u or large D_x in the discrete case, and for large n_x and n_u in the LQT case. Nevertheless, a benefit of Method 2 is that it can be run in parallel with the backward pass.

Although the above analysis results give a guideline for performance in large number of processors (computational cores), with finite number of processors, we can expect worse performance as we cannot allocate a single task to single processor. However, at the time of writing the typical number of cores in a GPU was already $\sim$10 k, and, therefore, the above analysis can be expected to become more and more accurate in the future with the steadily increasing number of computational cores.

C. Block Processing

Up to now, we have considered parallelization of control problems at a single time step level. That is, we initialize the element a_k for all k using (22) and then apply the parallel scan algorithm. Another option is to apply the parallel scan algorithm to nonoverlapping blocks of B time steps. That is, we can initialize the elements of the parallel scan with $V_{k\to k+B}$ for $k = S + nB$, with $n = 0, 1, \ldots, T/B - 1$. The initialization of each element can be done using (17) in $B - 1$ sequential steps. Then, we can apply the parallel scan algorithm to fuse the information from all blocks. In this case, the time complexity in a PRAM computer with large enough number of processors is $O(B + \log(N/B))$, so it is optimal to choose $B = 1$. Nevertheless, this approach can be useful if we have a limited number of processors.

In the case of LQT, an efficient implementation of the above can be achieved by using partial condensing [14], [15], where the idea is to reformulate the problem in terms of blocks of states and controls of size B

$$\bar{x}_{k/B} = \begin{bmatrix} x_k \\ \vdots \\ x_{k+B-1} \end{bmatrix}, \quad \bar{u}_{k/B} = \begin{bmatrix} u_k \\ \vdots \\ u_{k+B-1} \end{bmatrix}. \tag{67}$$

By elimination of the state variables inside each block, we can reformulate the problem in terms of the initial states of blocks only, $x_0, x_B, x_{2B}, \ldots$, which reduces the state dimensions from full state blocks to the original state dimension. The resulting problem is still an LQT problem, but with modified states and inputs, and hence we can use the proposed parallel LQT algorithms to solve it.

VI. Experimental Results

In this section, we experimentally evaluate the performance of the methods in simulated applications. We implemented the methods using the open-source TensorFlow 2.6 software library [26] using its Python 3.8 interface, which provides means to run parallel vectorized operations and parallel associative scans on GPUs. The experiments were run using NVIDIA A100-SXM GPU with 80 GB of memory. In the experiments, we concentrate on the computational speed benefits because the sequential and parallel version of the algorithm compute exactly the same solution (provided that it is unique), the only difference being in the computational speed. The computation speeds were measured by averaging over 10 runs and the time taken to (jit) compile the code was not included in the measurement.

We would like to point out that, as we are using TensorFlow with GPUs, the matrix operations on the individual time steps of the sequential algorithms are parallelized. Therefore, the sequential LQT algorithms can be interpreted as a TensorFlow parallel implementation of the Riccati recursion [13].

A. Experiment With Basic LQT

The aim of the experiment is to demonstrate the benefit of the proposed parallelization method over the classical sequential solution in an LQT problem. We consider a 2-D tracking problem obeying Newton's law [2, Example 4.4.2]. In this LQT problem, the aim is to steer an object to follow a given trajectory of reference points in 2-D by using applied forces as the control signals.

The state consists of the positions and velocities $x = \begin{bmatrix} p_x & p_y & v_x & v_y \end{bmatrix}^\top$ and the control signal $u = \begin{bmatrix} a_x & a_y \end{bmatrix}^\top$ contains the accelerations (forces divided by the mass which is unity in our case). If we assume that the control signal is kept fixed over each discretization interval $[t_k, t_{k+1}]$, then the dynamic model can be written as

$$x_{k+1} = F_k\, x_k + L_k\, u_k \tag{68}$$

where

$$F_k = \begin{bmatrix} 1 & 0 & \Delta t_k & 0 \\ 0 & 1 & 0 & \Delta t_k \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, L_k = \begin{bmatrix} \Delta t_k^2/2 & 0 \\ 0 & \Delta t_k^2/2 \\ \Delta t_k & 0 \\ 0 & \Delta t_k \end{bmatrix} \tag{69}$$

and $\Delta t_k = t_{k+1} - t_k$ is the sampling interval. Fig. 1 illustrates the scenario.

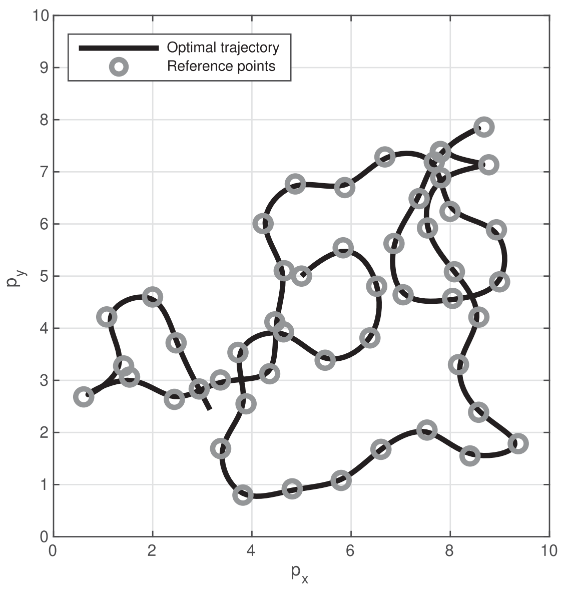

Fig. 1. Simulated trajectory from the linear control problem and optimal trajectory produced by LQT (see Section VI-A). The trajectory starts at (5,5).

The dynamic trajectory is discretized so that we add 10 intermediate steps between each of the reference point time steps which then results in a total of T times steps (giving $\Delta t_k = 0.1$). The cost function parameters are for $k = 0, \ldots, T-1$ selected to be

$$H_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, X_k = \begin{bmatrix} c_k & 0 \\ 0 & c_k \end{bmatrix}, U_k = 10^{-1}\, I_{2\times 2} \tag{70}$$

where $c_k = 100$ when there is a reference point at step k and 10^{-6} otherwise. At the final step we set $H_T = I_{4\times 4}$ and $X_T = I_{4\times 4}$. The reference trajectory contains the actual reference points at every 10th step k, and the intermediate steps are set equal to the previous reference point. At the final step the reference velocity is also zero. Together with the value $c_k = 10^{-6}$ at these intermediate points this results in tiny regularization of the intermediate paths, but the effect on the final result is practically negligible. It would also be possible to put $c_k = 0$ for the intermediate steps to yield almost the same result.

The results computing the control law (i.e, the backward pass) of the classic sequential LQT and the proposed parallel LQT on the GPU for $T = 10^2, \ldots, 10^5$ are shown in Fig. 2. The figure shows the run times of both in log–log scale on the top-right figure and the parallel result up to 10^4 is shown in linear scale on the top right. The speed up, computed as the ratio of sequential and parallel run times, is shown in the bottom figure. It can be seen that the parallel version is significantly faster than the sequential version (illustrated in the top left figure) and the logarithmic scaling of the parallel algorithm can also be seen (illustrated in the top right figure). The speed up (illustrated in the bottom figure) is of the order of $\sim$470 with 10^5 time points and although it is close to saturating, it still increases a bit.

We also ran the combined control law computation pass and the trajectory recovery pass on GPU, and the results are shown in Fig. 3. The advantage of parallel version over the sequential

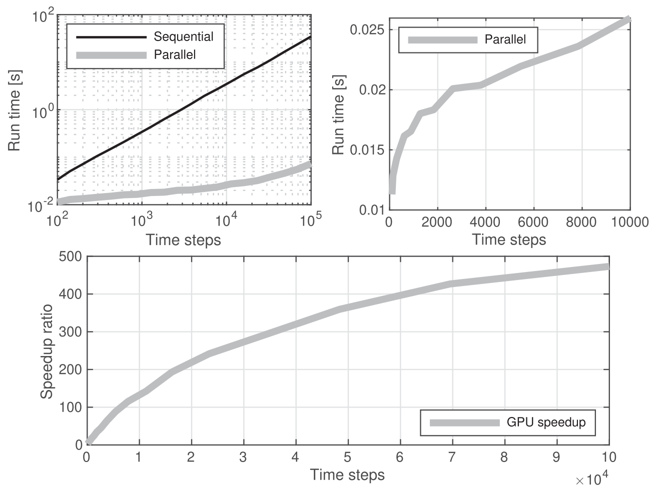

Fig. 2. LQT control law computation run times on GPU. The sequential and parallel run times are shown in the top left figure, and a zoom to the parallel run time is shown in the top right figure. The speed-up provided by the parallelization is shown in figure at the bottom.

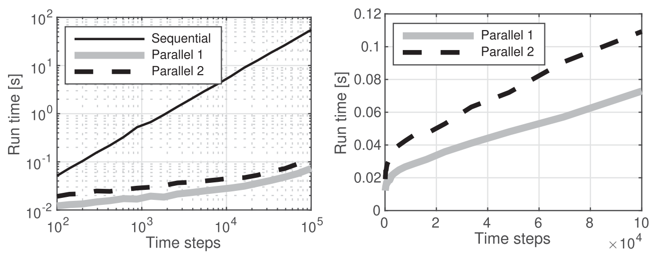

Fig. 3. GPU run times (left) and zoomed run times of the parallel methods (right) for combined control law computation and trajectory recovery (Methods 1 & 2) in LQT.

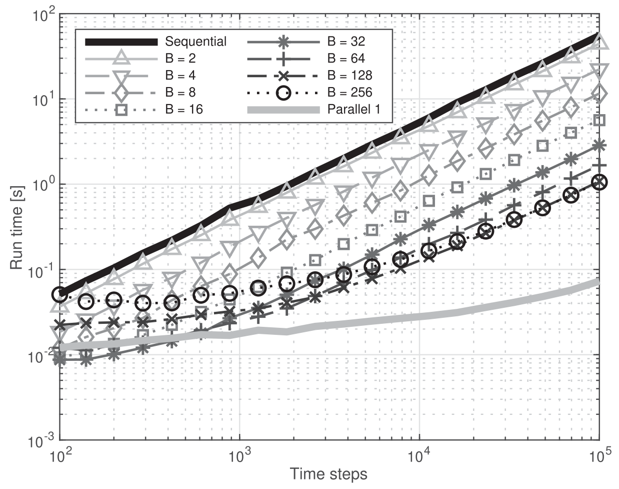

Fig. 4. Result of partial condensing with different values of block size B when the LQT is implemented by using parallel matrix operations within sequential Riccati solution and trajectory reconstruction.

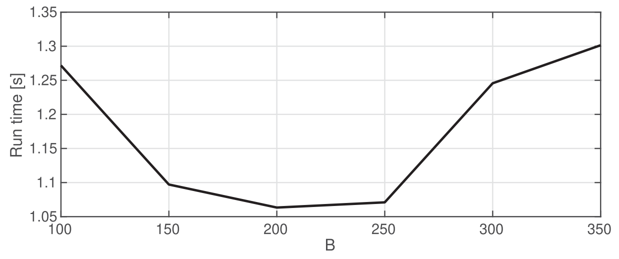

Fig. 5. Run times of partial condensing for $T = 10^5$ with different values of block size B when the LQT is implemented by using parallel matrix operations within sequential Riccati solution and trajectory reconstruction.

version can again be clearly seen. Furthermore, in this case, Method 1 for the parallel trajectory recovery is faster than Method 2.

B. Experiment With Partial Condensing

In this experiment we compare the proposed parallelization method for LQT to partial condensing [14], [15] based parallelization. We also demonstrate how parallel condensing can be combined with the proposed methodology to yield improved results. The same 2-D tracking problem and data are used as in Section VI-A.

As discussed in Section V-C, the idea of partial condensing is to reduce a control problem of length T to a control problem of length T/B by dividing the problem into blocks of length B. In each of these blocks we can eliminate all but the state at the beginning of the block, which effectively reduces a control problem with state dimension n_x, input dimension $B\,n_u$, and length T/B. The required computations are 1) conversion of the model into block form, 2) computation of LQT solution of length T/B, and 3) reconstruction of the intermediate states in each block. Partial condensing allows for parallelization of the computations because steps 1) and 3) are fully parallelizable and step 2) can be efficiently implemented by using parallel matrix operations within the sequential Riccati solution and trajectory reconstruction [13]–[15].

The GPU run times of the aforementioned parallel condensing method with $B = 2, 4, 8, \ldots, 256$ are shown Fig. 4. The run times of the classical backward–forward sequential LQT solution for trajectory recovery and of the proposed parallel method with trajectory recovery with Method 1 are also shown in the figure. The trajectory lengths were $T = 10^2, \ldots, 10^5$. It can be seen that the run times of partial condensing methods are significantly lower than of the classical sequential LQT while still, for the most of the cases, higher than of the proposed parallel method. It can be seen that with short trajectory lengths the partial condensing method is faster than the proposed parallel method when $B = 16$ or 32. However, with larger trajectory lengths the proposed parallel method is faster.

In the results of Fig. 4 we can see some evidence of an effect that when increasing B, the run time no longer decreases after a certain value. This is confirmed in Fig. 5 which shows the run times of the partial condensing over trajectory of length $T = 10^5$ with different values of B. It can be seen that the run time attains minimum somewhere around $B = 200$ and after that the run time starts to increase.

As discussed in Section V-C, it is also possible to combine partial condensing with the proposed parallelization methodology. This can be done by implementing the LQT solution

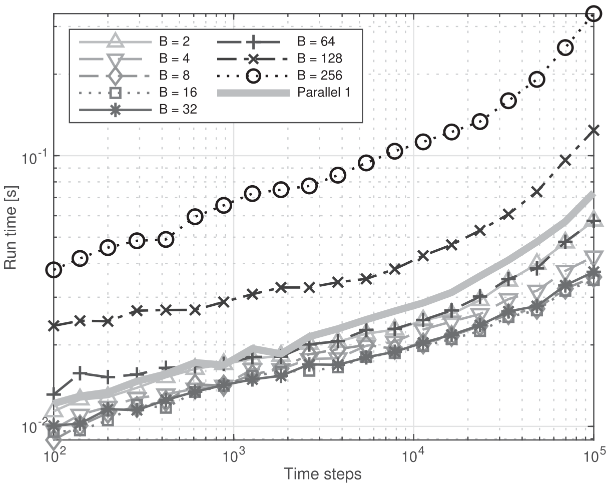

Fig. 6. Result of partial condensing with different values of N_c when the LQT is implemented by using the proposed parallel methods.

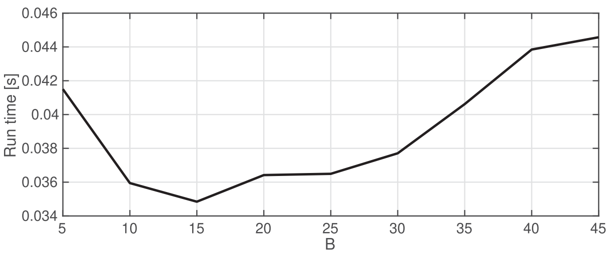

Fig. 7. Run times of partial condensing for $T = 10^5$ with different values of B when the LQT is implemented by using the proposed parallel methods.

(of length T/B) by using one of the parallel methods. Fig. 6 shows the results of using this combined approach. It can be seen that the partial condensing can be used to improve run time of the proposed parallel methods when B is in suitable range (2–64 in this case). When B is too large (128 or 256), then partial condensing no longer improves the run times. This also happens with $B = 64$ when the trajectory length is short. Fig. 7 shows the run times with $T = 10^5$ as function of B. It can be seen in the figure that the minimum run time is attained roughly at value $B = 15$.

C. Experiment With Increasing State Dimensionality

In this experiment the aim is to test the scaling of run time when the dimensionality of the state increases. For this purpose, we use a slight modification of a mass-spring-damper problem [42] [43, Sec. 2.5.3], where we have removed the control constraints, as we do not consider them in this article. This is a linear model for controlling a chain of N masses $m = 1$ kg connected with springs with constants $c = 1$ kg/s^2 and dampers with constants $d = 0.2$ kg/s. The control is applied to the first and last mass. The model is thus (see Fig. 8)

$$\ddot{y}_1 = \frac{c}{m}\left[-2y_1 + y_2\right] + \frac{d}{m}\left[-2\dot{y}_1 + \dot{y}_2\right] + \frac{1}{m}\,u_1$$

$$\ddot{y}_i = \frac{c}{m}\left[y_{i-1} - 2y_i + y_{i+1}\right] + \frac{d}{m}\left[\dot{y}_{i+1} - 2\dot{y}_i + \dot{y}_{i+1}\right]$$

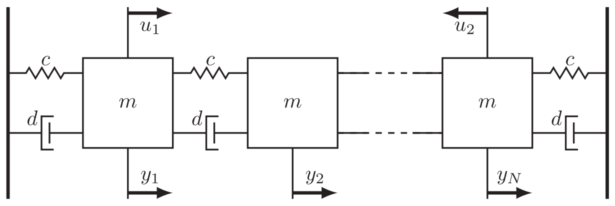

Fig. 8. Illustration of the mass-spring-damper problem (see Section VI-C).

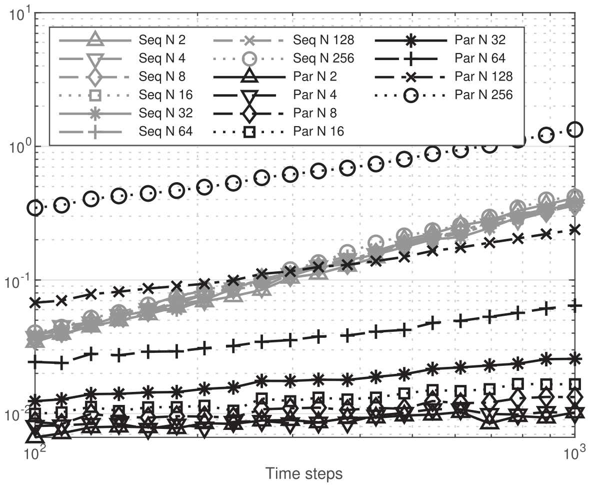

Fig. 9. Run times of the mass-spring-damper problem with different number of masses N corresponding to state dimensionalities $2N$.

$$\ddot{y}_N = \frac{c}{m}\left[-2y_N + y_{N-1}\right] + \frac{d}{m}\left[-2\dot{y}_N + \dot{y}_{N-1}\right] - \frac{1}{m}\,u_2. \tag{71}$$

where $i = 2, \ldots, N-1$. The state of the system is $x = \begin{bmatrix} y_1 & \dot{y}_1 & \cdots & y_N & \dot{y}_N \end{bmatrix}^\top$. Similar to the case in [43], the aim is to control the system to origin from an initial condition, where the first mass and middle mass, with index $i = \lfloor \frac{N}{2} \rfloor + 1$, are started at position 1 m. The model is uniformly discretized, with closed-form zero-order-hold (ZOH) discretization, using varying number of time steps $T = 10^2, \ldots, 10^3$ such that the total control interval length is 10 s.

The cost function is

$$C[u_{0:T}] = \frac{1}{2} x_T^\top X_T x_T + \frac{1}{2}\sum_{n=0}^{T-1} x_n^\top X x_n + \frac{1}{2}\sum_{n=0}^{T-1} u_n^\top U u_n \tag{72}$$

with $X = X_T = I$ and $U = 0.1I$.

Fig. 9 shows the results of sequential LQT and proposed parallel LQT. Due to parallelization of the matrix operations on the individual steps of the sequential LQT, its run time is essentially independent of the state dimension. The parallel method, however, experiences significant run time increase with larger state dimension. Although when the number of masses N is 2–64, the run times of the parallel methods are shorter than those of the sequential method, with $N = 128$ the parallel method is slower with small numbers of time steps and with $N = 256$ it is slower with all the time step counts.

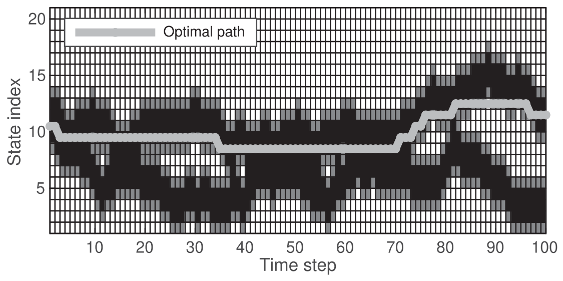

Fig. 10. Finite state space scenario where the aim is to find a minimum cost path from left to right by steering up or down (see Section VI-D). The grayscale values show the cost function values.

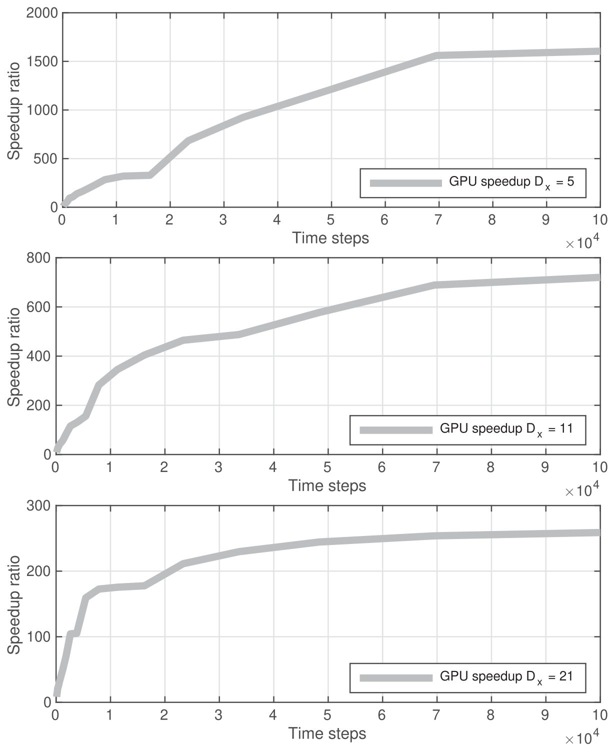

Fig. 11. Finite state-space GPU speed- ps for control law computation (parallel verses sequential) with state dimensions 5, 11, and 21.

D. Experiment With Finite State Space

In this experiment, we consider an aircraft routing problem [2, Example 6.1.1], where an aircraft proceeds from left to right, and the aim is to control up and down movement on a finite grid so that the total cost is minimized. Each of the grid points incurs a cost $\{0, 1, 2\}$ which is related to the fuel required to go through it. Taking a control up or down costs a single unit, and proceeding straight costs nothing. The scenario is illustrated in Fig. 10.

In this case, we tested the finite-state control law computation using different state dimensionalities as the parallel combination rule can be expected to have a dependence on the state dimensionality when the number of computational cores is limited. The GPU speed ups for state dimensions $D_x \in \{5, 11, 21\}$ are shown in Fig. 11. It can be seen that with state dimensionality $D_x = 5$ the speed up reaches $\sim$1500 with $T = 10^5$ and is still slightly increasing. With the state dimensionality $D_x = 11$ the maximum achieved speed up is roughly $\sim$700, and with state dimensionality $D_x = 21$ the speed up saturates to a value around 350. However, with all of the state dimensionalities parallelization provides a significant speed up.

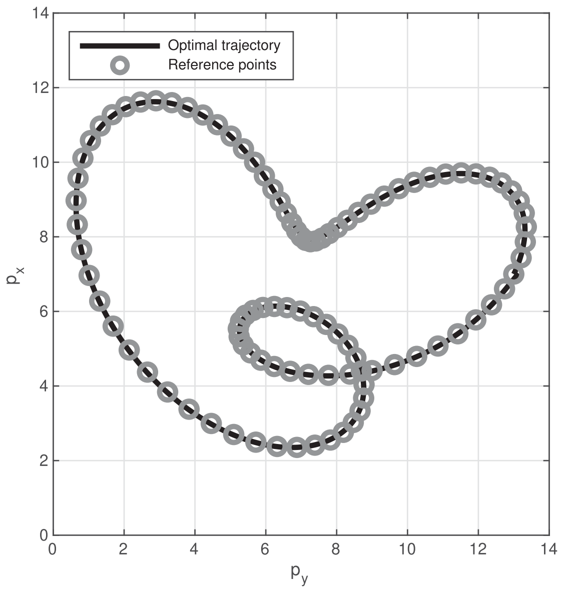

Fig. 12. Simulated trajectory from the nonlinear control problem and optimal trajectory produced by nonlinear LQT (see Section VI-E).

E. Experiment With Nonlinear LQT

This experiment is concerned with a nonlinear dynamic model, where we control a simple unicycle [34, Sec. 13.2.4.1] whose state consists of 2-D position, orientation, and speed $x = \begin{bmatrix} p_x & p_y & \theta & s \end{bmatrix}^\top$. The aim is to steer the device to follow a given position and orientation trajectory which corresponds to going around a fixed race track multiple times. The control signal consists of the tangential acceleration and turn rate $u = \begin{bmatrix} a & \omega \end{bmatrix}^\top$. The discretized nonlinear model has the form

$$x_{k+1} = f_k(x_k, u_k) \tag{73}$$

where

$$f_k(x, u) = \begin{bmatrix} p_x + s\,\cos(\theta)\,\Delta t_k \\ p_y + s\,\sin(\theta)\,\Delta t_k \\ \theta + \omega\,\Delta t_k \\ s + a\,\Delta t_k \end{bmatrix} \tag{74}$$

and $\Delta t_k = t_{k+1} - t_k$. Fig. 12 shows the trajectory and the optimal trajectory produced by the nonlinear LQT.

The cost function parameters were selected to be the following for $k = 0, \dots, T-1$:

$$H_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

$$X_k = \begin{bmatrix} c_k & 0 & 0 \\ 0 & c_k & 0 \\ 0 & 0 & d_k \end{bmatrix}, U_k = \begin{bmatrix} 1 & 0 \\ 0 & 100 \end{bmatrix} \tag{75}$$

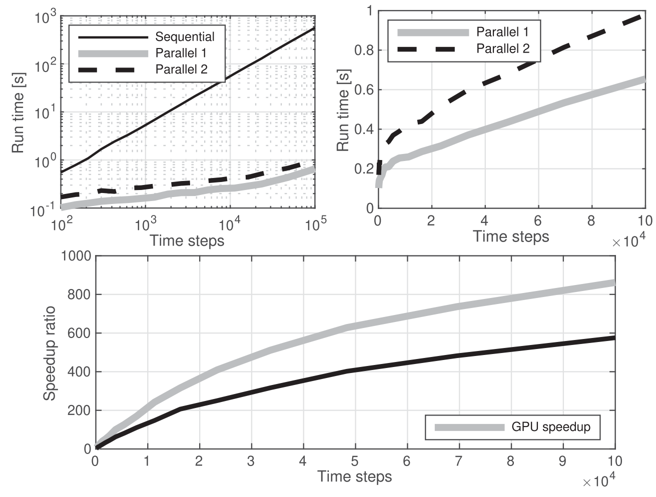

Fig. 13. Nonlinear LQT GPU run times and speedup for 10 iterations.

where $c_k = 100, d_k = 1000$, when there is a reference point at step k, and 10^{-6} otherwise. The latter values were also used for the terminal step $k = T$. The time step length was $\Delta t_k = 0.1$. An iterated nonlinear LQT using a Taylor series approximation was applied to the model, and the number of iterations was fixed to 10. Fig. 13 shows the run times for GPU. It can be seen that parallelization provides a significant speed up over sequential computation. When the Method 1 was used to compute the recovered trajectory at each iteration step, the speed up grows to around 800 for $T = 10^5$ on GPU. Method 2 reaches a speed up of around 600.

VII. Conclusion

In this article, we have shown how dynamic programming solutions to optimal control problems and their linear quadratic special case, the LQT, can be parallelized in the temporal domain by defining the corresponding associative operators and making use of parallel scans. The parallel methods have logarithmic complexity with respect to the number of time steps, which significantly reduces the linear complexity of standard (sequential) methods for long time horizon control problems. These benefits are shown via numerical experiments run on a GPU. This article shows that the contribution is timely as it can leverage modern hardware and software for parallel computing, such as GPUs and TensorFlow.

An interesting future extension of the framework would be parallel stochastic dynamic programming solution to stochastic control problems [1], [35]. As discussed in Section IV-D, this is straightforward in the LQT case due to certainty equivalence, but the general stochastic case is not straightforward. Formally it is possible to replace the state x_k with the distribution of the state p_k and consider condition value functionals of the form $V_{i\to j}[p_i, p_j]$ and value functionals of the form $V_i[p_i]$. The present framework then, in principle, applies as such. In particular, when the distributions have finite-dimensional sufficient statistics, this can lead to tractable methods. Unfortunately, unlike in the sequential dynamic programming case, more generally, this approach does not seem to lead to a tractable algorithm.

Another interesting extension is to consider continuous optimal control problems in which case also the stochastic control solution has certain group properties [44] which might allow for parallelization. However, the benefit of parallelization in the continuous case is not as clear as in discrete-time case because of the infinite number of time steps.

Appendix A
Derivations and Proofs

A. Proof of Theorem 2

In this Appendix we prove Theorem 2. We first prove (18). From (1) and (2), we obtain

$$\begin{aligned}
V_k(x_k) &= \min_{u_{k:T-1}} \ell_T(x_T) + \sum_{n=k}^{T-1} \ell_n(x_n,u_n) \\
&= \min_{u_{k:T-1}} \sum_{n=k}^{i-1} \ell_n(x_n,u_n) + \sum_{n=i}^{T-1} \ell_n(x_n,u_n) + \ell_T(x_T) \\
&= \min_{u_{k:i-1}} \sum_{n=k}^{i-1} \ell_n(x_n,u_n) \\
&\quad + \min_{u_{i:T-1}} \left[\sum_{n=i}^{T-1} \ell_n(x_n,u_n) + \ell_T(x_T) \right] \\
&= \min_{u_{k:i-1}} \sum_{n=k}^{i-1} \ell_n(x_n,u_n) + V_i(x_i) \qquad (76)
\end{aligned}$$

where the previous minimizations are subject to the trajectory constraints in (1).

We can also minimize over x_i explicitly such that

$$\begin{aligned}
V_k(x_k) &= \min_{x_i} \left[\min_{u_{k:i-1}} \left[\sum_{n=k}^{i-1} \ell_n(x_n,u_n) \right] + V_i(x_i) \right] \qquad (77) \\
&= \min_{u_{i-1}} \left[V_{k\to i}(x_k,x_i) + V_i(x_i) \right] \qquad (78)
\end{aligned}$$

which proves (18).

Proceeding analogously, we now prove (17). From (15), we obtain

$$\begin{aligned}
V_{k\to i}(x_k,x_i) &= \min_{u_{k:i-1}} \sum_{n=k}^{i-1} \ell_n(x_n,u_n) \\
&= \min_{u_{k:j-1}} \left[\sum_{n=k}^{j-1} L_n(x_n,u_n) + \min_{u_{j:i-1}} \sum_{n=j}^{i-1} \ell_n(x_n,u_n) \right] \\
&= \min_{u_{k:j-1}} \left[\sum_{n=k}^{j-1} \ell_n(x_n,u_n) + V_{j\to i}(x_j,x_i) \right] \\
&= \min_{x_j} \min_{u_{k:j-1}} \left[\sum_{n=k}^{j-1} \ell_n(x_n,u_n) + V_{j\to i}(x_j,x_i) \right] \\
&= \min_{x_j} \left[\min_{u_{k:j-1}} \left(\sum_{n=k}^{j-1} \ell_n(x_n,u_n) \right) + V_{j\to i}(x_j,x_i) \right] \\
&= \min_{x_j} \left[V_{k\to j}(x_k,x_j) + V_{j\to i}(x_j,x_i) \right] \qquad (79)
\end{aligned}$$

which completes the proof of (15).

B. Proof of LQT Combination Rule

In this Appendix, we prove the combination rule for LQT in Lemma 10. Combining $V_{k\to j}(x_k, x_j)$ and $V_{j\to i}(x_j, x_i)$ of the form (32), we obtain

$$\begin{aligned}
&V_{k\to i}(x_k, x_i)\\
&= \min_{x_j}\left\{\max_{\lambda_1} g_{k\to j}(\lambda_1; x_k, x_j) + \max_{\lambda_2} g_{j\to i}(\lambda_2; x_j, x_i)\right\}\\
&= \mathrm{z} + \max_{\lambda_1,\lambda_2}\min_{x_j}\left\{\frac{1}{2}x_k^\top J_{k,j}x_k - x_k^\top \eta_{k,j} - \frac{1}{2}\lambda_1^\top C_{k,j}\lambda_1\right.\\
&\quad - \lambda_1^\top\left(x_j - A_{k,j}x_k - b_{k,j}\right) + \frac{1}{2}x_j^\top J_{j,i}x_j - x_j^\top \eta_{j,i}\\
&\quad \left.-\frac{1}{2}\lambda_2^\top C_{j,i}\lambda_2 - \lambda_2^\top\left(x_i - A_{j,i}x_j - b_{j,i}\right)\right\}.
\end{aligned}$$

We prove the result by calculating the minimum w.r.t. x_j and maximum w.r.t. λ_1, leaving the Lagrange multiplier λ_2 as the Lagrange multiplier of $V_{k\to i}(x_k, x_i)$.

Setting the gradient w.r.t. x_j equal to zero, we obtain

$$J_{j,i}x_j = \lambda_1 + \eta_{j,i} - A_{j,i}^\top\lambda_2 \tag{80}$$

where $J_{j,i}$ is not invertible in general.

Setting the gradient w.r.t λ_1 equal to zero, we have

$$x_j = -C_{k,j}\lambda_1 + A_{k,j}x_k + b_{k,j}. \tag{81}$$

Then, substituting (81) into (80) yields

$$\begin{aligned}
\lambda_1 &= \left(I + J_{j,i}C_{k,j}\right)^{-1}\\
&\quad\left[-\eta_{j,i} + A_{j,i}^\top\lambda_2 + J_{j,i}\left(A_{k,j}x_k + b_{k,j}\right)\right].
\end{aligned} \tag{82}$$

Substituting (82) into (81), we obtain

$$\begin{aligned}
x_j = &-C_{k,j}\left(I + J_{j,i}C_{k,j}\right)^{-1}A_{j,i}^\top\lambda_2\\
&- C_{k,j}\left(I + J_{j,i}C_{k,j}\right)^{-1}\left[-\eta_{j,i} + J_{j,i}\left(A_{k,j}x_k + b_{k,j}\right)\right]\\
&+ A_{k,j}x_k + b_{k,j}.
\end{aligned} \tag{83}$$

We now substitute the stationary points (82) and (83) in each of the terms in $V_{k\to i}(x_k, x_i)$ to recover a function of the form (32). This step involves the use of long mathematical expressions so it is left out of the article. Lemma 10 then follows by term identification.

C. Proof of Parallel LQT

In this section, we prove Lemma 11.

1) Proof of (40): We first show the form of $V_{k\to k+1}(x_k, x_{k+1})$ in its dual representation for $k = S, \ldots, T$. Using (15) and the LQT problem formulation in (5), we obtain

$$\begin{aligned}
V_{k\to k+1}(x_k, x_{k+1}) &= \min_{u_k}\ell_k(x_k, u_k)\\
&= \min_{u_k}\frac{1}{2}(H_kx_k - r_k)^\top X_k(H_kx_k - r_k)\\
&\quad + \frac{1}{2}u_k^\top U_ku_k\\
&= \mathrm{z}+\min_{u_k}\frac{1}{2}x_k^\top H_k^\top X_kH_kx_k - x_k^\top H_k^\top X_kr_k\\
&\quad + \frac{1}{2}u_k^\top U_ku_k
\end{aligned} \tag{84}$$

subject to

$$x_{k+1} = F_kx_k + c_k + L_ku_k. \tag{85}$$

The Lagrangian of $V_{k\to k+1}(x_k, x_{k+1})$ is [24]

$$\begin{aligned}
&L_{k\to k+1}\left(u_k, \lambda; x_k, x_{k+1}\right)\\
&= \mathrm{z} + \frac{1}{2}x_k^\top H_k^\top X_kH_kx_k - x_k^\top H_k^\top X_kr_k\\
&\quad + \frac{1}{2}u_k^\top U_ku_k + \lambda^\top\left(L_ku_k - \left(x_{k+1} - F_kx_k - c_k\right)\right)
\end{aligned}$$

and the dual function is [24]

$$\begin{aligned}
&g_{k\to k+1}\left(\lambda; x_k, x_{k+1}\right)\\
&= \min_{u_k} L_{k\to k+1}\left(u_k, \lambda; x_k, x_{k+1}\right)\\
&= \mathrm{z} + \frac{1}{2}x_k^\top H_k^\top X_kH_kx_k - x_k^\top H_k^\top X_kr_k\\
&\quad - \frac{1}{2}\lambda^\top L_kU_k^{-1}L_k^\top\lambda - \lambda^\top\left(x_{k+1} - F_kx_k - c_k\right)
\end{aligned} \tag{86}$$

where the minimum is obtained setting the gradient of $L_{k\to k+1}(\cdot)$ w.r.t. u_k equal to zero, which gives

$$u_k = -U_k^{-1}L_k\lambda. \tag{87}$$

Comparing (86) with (33) proves the initialization in (38). Then, by applying Theorem 2, which is equivalent to Lemma 10 in the LQT setting, we complete the proof of (40).

2) Proof of (41): We use induction backward to prove (41). At the last time step, from Lemma 11, we have

$$\begin{aligned}
&V_{T\to T+1}(x_T, x_{T+1})\\
&= \max_{\lambda}\left[\mathrm{z} + \frac{1}{2}x_T^\top H_T^\top X_TH_Tx_T - x_T^\top H_T^\top r_T - \lambda^\top x_{T+1}\right].
\end{aligned}$$

For $x_{T+1} \neq 0$, this function is infinite. For $x_{T+1} = 0$, we have

$$V_{T\to T+1}(x_T, 0) = \frac{1}{2}x_T^\top H_T^\top X_TH_Tx_T - x_T^\top H_T^\top r_T$$

which coincides with $V_T(x_T)$, see Section II-B.

We now assume that (41) holds for $k+1$, which implies that we have

$$\begin{aligned}
A_{k+1,T+1} &= 0\\
b_{k+1,T+1} &= 0\\
C_{k+1,T+1} &= 0\\
\eta_{k+1,T+1} &= v_{k+1}\\
J_{k+1,T+1} &= S_{k+1}
\end{aligned}$$

where v_{k+1} and S_{k+1} are the parameters in (6), and then show that (41) holds for k. From Lemma 11, we have

$$\begin{aligned}
A_{k,k+1} &= F_k\\
b_{k,k+1} &= c_k\\
C_{k,k+1} &= L_kU_k^{-1}L_k^\top\\
\eta_{k,k+1} &= H_k^\top X_kr_k\\
J_{k,k+1} &= H_k^\top X_kH_k.
\end{aligned}$$

By applying the combination rules in Lemma 10, we obtain

$$\begin{aligned}A_{k,T+1} &= 0\\ b_{k,T+1} &= 0\\ C_{k,T+1} &= 0\\ \eta_{k,T+1} &= F_k^\top (I + S_{k+1} L_k U_k^{-1} L_k^\top)^{-1} (v_{k+1} - S_{k+1} c_k)\\ &\quad + H_k^\top X_k r_k\\ J_{k,T+1} &= F_k^\top (I + S_{k+1} L_k U_k^{-1} L_k^\top)^{-1} S_{k+1} F_k + H_k^\top X_k H_k.\end{aligned} \tag{88}$$

We need to prove that these equations are equivalent to (7) and (8). We first prove that $\eta_{k,T+1} = v_k$, which requires proving that the following identity holds:

$$F_k^\top (I + S_{k+1} L_k U_k^{-1} L_k^\top)^{-1} = \left(F_k^\top - L_k K_k\right)^\top. \tag{89}$$

On one hand, the right-hand side can be written as

$$\begin{aligned}&\left(F_k^\top - L_k K_k\right)^\top\\ &= F_k - F_k^\top S_{k+1} L_k \left(L_k^\top S_{k+1} L_k + U_k\right)^{-1} L_k^\top.\end{aligned} \tag{90}$$

On the other hand, by applying the matrix inversion lemma, the left-hand side becomes

$$\begin{aligned}&F_k^\top (I + S_{k+1} L_k U_k^{-1} L_k^\top)^{-1}\\ &= F_k^\top \left(I - S_{k+1} L_k \left(U_k + L_k^\top S_{k+1} L_k\right)^{-1} L_k^\top\right)\end{aligned} \tag{91}$$

which proves $\eta_{k,T+1} = v_k$.

To prove that $J_{k,T+1} = S_k$, we need to prove that

$$F_k^\top (I + S_{k+1} L_k U_k^{-1} L_k^\top)^{-1} S_{k+1} F_k = F_k^\top S_{k+1} (F_k - L_k K_k). \tag{92}$$

On one hand, the right-hand side can be written as

$$\begin{aligned}&F_k^\top S_{k+1} (F_k - L_k K_k)\\ &= F_k^\top S_{k+1} \left(F_k - L_k \left(L_k^\top S_{k+1} L_k + U_k\right)^{-1} L_k^\top S_{k+1} F_k\right)\\ &= F_k^\top S_{k+1} F_k - F_k^\top S_{k+1} L_k \left(L_k^\top S_{k+1} L_k + U_k\right)^{-1} L_k^\top S_{k+1} F_k.\end{aligned} \tag{93}$$

On the other hand, using (91), the left-hand side is

$$\begin{aligned}&F_k^\top (I + S_{k+1} L_k U_k^{-1} L_k^\top)^{-1} S_{k+1} F_k\\ &= F_k^\top \left(I - S_{k+1} L_k \left(U_k + L_k^\top S_{k+1} L_k\right)^{-1} L_k^\top\right) S_{k+1} F_k\\ &= F_k^\top S_{k+1} F_k - F_k^\top S_{k+1} L_k \left(U_k + L_k^\top S_{k+1} L_k\right)^{-1} L_k^\top S_{k+1} F_k\end{aligned} \tag{94}$$

which proves the result.

D. Proof of Optimal Trajectory Recovery

In this Appendix, we prove Lemma 15. Substituting $V_k(x_k)$ of the form (6) and $V_{S\to k}(x_S, x_k)$ of the form (32) into (31), we obtain

$$\begin{aligned}x_k^* &= \arg\min_{x_k} \max_{\lambda} \frac{1}{2} x_S^\top J_{S,k} x_S - x_S^\top \eta_{S,k}\\ &\quad - \frac{1}{2} \lambda^\top C_{S,k} \lambda - \lambda^\top \left(x_k - A_{S,k} x_S - b_{S,k}\right)\\ &\quad + \frac{1}{2} x_k^\top S_k x_k - v_k^\top x_k.\end{aligned} \tag{95}$$

The minimum of (95) w.r.t. x_k can be found by setting the gradient of the function equal to zero, which yields

$$x_k = S_k^{-1} \left(\lambda + v_k\right). \tag{96}$$

We substitute (96) into the function (without argmin) in (95) to obtain

$$\begin{aligned}&\max_{\lambda} \frac{1}{2} x_S^\top J_{S,k} x_S - x_S^\top \eta_{S,k} - \frac{1}{2} \lambda^\top \left(C_{S,k} + S_k^{-1}\right) \lambda\\ &+ v_k^\top S_k^{-1} v_k - \lambda^\top \left(S_k^{-1} v_k - A_{S,k} x_S - b_{S,k}\right).\end{aligned}$$

Making the gradient of this function w.r.t. λ equal to zero, we obtain that the maximum is obtained for

$$\lambda = \left(C_{S,k} + S_k^{-1}\right)^{-1} \left(-S_k^{-1} v_k + A_{S,k} x_S + b_{S,k}\right). \tag{97}$$

Substituting (97) into (96), we obtain (55), which finishes the proof of Lemma 15.

Acknowledgment

The authors would like to thank A. Corenflos for help in TensorFlow programming.

References

[1] R. F. Stengel, *Optimal Control and Estimation*. New York, NY, USA: Dover publications, 1994.

[2] F. L. Lewis and V. L. Syrmos, *Optimal Control*. 2nd ed. Hoboken, NJ, USA: Wiley, 1995.

[3] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Belmont, MA, USA: Athena Scientific, 2005.

[4] J. Biggs and W. Holderbaum, "Optimal kinematic control of an autonomous underwater vehicle," *IEEE Trans. Autom. Control*, vol. 54, no. 7, pp. 1623–1626, Jul. 2009.

[5] A. Komaee and A. Bensoussan, "Optimal control of hidden Markov models with binary observations," *IEEE Trans. Autom. Control*, vol. 59, no. 1, pp. 64–77, Jan. 2014.

[6] G. Zhao and M. Zhu, "Pareto optimal multirobot motion planning," *IEEE Trans. Autom. Control*, vol. 66, no. 9, pp. 3984–999, Sep. 2021.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[8] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton Univ. Press, 1957.

[9] R. E. Bellman and S. E. Dreyfus, "Applied dynamic programming" RAND Corp., Santa Monica, CA, USA, Tech. Rep. R-352-PR, 1962.

[10] A. Pakniyat and P. E. Caines, "On the relation between the minimum principle and dynamic programming for classical and hybrid control systems," *IEEE Trans. Autom. Control*, vol. 62, no. 9, pp. 4347–4362, Sep. 2017.

[11] M. Gengler, "An introduction to parallel dynamic programming," in *Solving Combinatorial Optimization Problems in Parallel*. Berlin, Germany: Springer, 1996, pp. 87–114.

[12] S. Dormido Canto, A. P. de Madrid, and S. D. Bencomo, "Parallel dynamic programming on clusters of workstations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 9, pp. 785–798, Sep. 2005.

[13] G. Frison and J. B. Jørgensen, "Parallel implementation of Riccati recursion for solving linear-quadratic control problems," in *Proc. 18th Nordic Process Control Workshop*, 2013, pp. 1–6.

[14] D. Axehill, "Controlling the level of sparsity in MPC," *Syst. Control Lett.*, vol. 76, pp. 1–7, Feb. 2015.

[15] G. Frison, D. Kouzoupis, J. B. Jørgensen, and M. Diehl, "An efficient implementation of partial condensing for nonlinear model predictive control," in *Proc. IEEE 55th Conf. Decis. Control*, 2016, pp. 4457–4462.

[16] J. Casti, M. Richardson, and R. Larson, "Dynamic programming and parallel computers," *J. Optim. Theory Appl.*, vol. 12, no. 4, pp. 423–438, 1973.

[17] S. J. Wright, "Partitioned dynamic programming for optimal control," *SIAM J. Optim.*, vol. 1, no. 4, pp. 620–642, 1991.

[18] J. V. Frasch, S. Sager, and M. A. Diehl, "A parallel quadratic programming method for dynamic optimization problems," *Math. Program. Computation*, vol. 7, pp. 289–329, 2015.
[19] S. Shin, T. Faulwasser, M. Zanon, and V. M. Zavala, "A parallel decomposition scheme for solving long-horizon optimal control problems," in *Proc. IEEE 58th Conf. Decis. Control*, 2019, pp. 5264–5271.
[20] Y. Jiang, J. Oravec, B. Houska, and M. Kvasnica, "Parallel MPC for linear systems with input constraints," *IEEE Trans. Autom. Control*, vol. 66, no. 7, pp. 3401–3408, Jul. 2021.
[21] J. L. Calvet and G. Viargues, "Invariant imbedding and parallelism in dynamic programming for feedback control," *J. Optim. Theory Appl.*, vol. 87, no. 1, pp. 121–140, 1995.
[22] G. E. Blelloch, "Scans as primitive parallel operations," *IEEE Trans. Comput.*, vol. 38, no. 11, pp. 1526–1538, Nov. 1989.
[23] G. E. Blelloch, "Prefix sums and their applications," School Comput. Science, Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-90-190, 1990.
[24] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[25] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *Proc. Int. Conf. Inform. Control, Automat. Robot.*, 2004, pp. 222–229.
[26] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," in *Proc. 12th USENIX Conf. Operating Syst. Des. Implementation*, 2015, pp. 265–283, [Online]. Available: https://www.tensorflow.org/
[27] S. Särkkä and A. F. García-Fernández, "Temporal parallelization of Bayesian smoothers," *IEEE Trans. Autom. Control*, vol. 66, pp. 299–306, Jan. 2021.
[28] F. Yaghoobi, A. Corenflos, S. Hassan, and S. Särkkä, "Parallel iterated extended and sigma-point Kalman smoothers," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process*, 2021, pp. 5350–5354.
[29] S. Hassan, S. Särkkä, and A. F. García-Fernández, "Temporal parallelization of inference in hidden Markov models," *IEEE Trans. Signal Process.*, vol. 69, pp. 4875–4887, 2021.
[30] P. M. Dower, W. M. McEneaney, and H. Zhang, "Max-plus fundamental solution semigroups for optimal control problems," in *Proc. SIAM Conf. Control Appl.*, 2015, pp. 368–375.
[31] H. Zhang and P. M. Dower, "Max-plus fundamental solution semigroups for a class of difference Riccati equations," *Automatica*, vol. 52, pp. 103–110, 2015.
[32] H. Zhang and P. M. Dower, "A max-plus based fundamental solution for a class of discrete time linear regulator problems," *Linear Algebra Appl.*, vol. 471, pp. 693–729, 2015.
[33] J. Xu, T. van den Boom, and B. De Schutter, "Model predictive control for stochastic max-plus linear systems with chance constraints," *IEEE Trans. Autom. Control*, vol. 64, no. 1, pp. 337–342, Jan. 2019.
[34] S. M. LaValle, *Planning Algorithms*. Cambridge U.K.: Cambridge Univ. Press, 2006.
[35] P. S. Maybeck, *Stochastic Models, Estimation and Control*. vol. 3. New York, NY, USA: Academic Press, 1982.
[36] T. Rauber and G. Rünger, *Parallel Programming: For Multicore and Cluster Systems*, 2nd ed. Berlin, Germany: Springer, 2013.
[37] G. Barlas, *Multicore and GPU Programming: An Integrated Approach*. San Mateo, CA, USA: Morgan Kaufmann, 2015.
[38] T. M. Apostol, *Calculus*. vol. 1. Hoboken, NJ, USA: Wiley, 1967.
[39] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, MA, USA: The MIT Press, 2009.
[40] S. Arora and B. Barak, *Computational Complexity: A. Modern Approach*. Cambridge U.K.: Cambridge Univ. Press, 2007.
[41] J. M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*. Berlin, Germany: Springer, 1988.
[42] T. Englert, A. Völz, F. Mesmer, S. Rhein, and K. Graichen, "A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC)," *Optim. Eng.*, vol. 20, pp. 769–809, 2019.
[43] B. Käpernick, "Gradient-based nonlinear model predictive control with constraint transformation for fast dynamical systems," Ph.D. dissertation, Fac. Eng., Comput. Sci. Psychol., Universität Ulm, Ulm, Germany, 2016.
[44] M. Nisio, *Stochastic Control Theory: Dynamic Programming Principle*. Berlin, Germany: Springer, 2015.

Simo Särkkä (Senior Member, IEEE) received the master's of science (Tech.) degree (with distinction) in engineering physics and mathematics, and the doctor's of science (Tech.) degree (with distinction) in electrical and communications engineering from the Helsinki University of Technology, Espoo, Finland, in 2000 and 2006, respectively.

He is currently an Associate Professor with Aalto University, Espoo, Finland, and an Adjunct Professor with Tampere University of Technology, Tampere, Finland, and Lappeenranta University of Technology, Lappeenranta, Finland. He has authored or coauthored over 150 peer-reviewed scientific articles and the books *Bayesian Filtering and Smoothing* (Cambridge University Press) and *Applied Stochastic Differential Equations* (Cambridge University Press) along with the Chinese translation of the former. His research interests are in multisensor data processing systems with applications in location sensing, health and medical technology, machine learning, inverse problems, and brain imaging.

Dr. Särkkä serves as a Senior Area Editor for IEEE SIGNAL PROCESSING LETTERS.

Ángel F. García-Fernández received the telecommunication engineering degree (with honours) and the Ph.D. degree from Universidad Politécnica de Madrid, Madrid, Spain, in 2007 and 2011, respectively.

He is currently a Lecturer with the Department of Electrical Engineering and Electronics, University of Liverpool, Liverpool, U.K. He previously held postdoctoral positions with Universidad Politécnica de Madrid, Madrid, Spain; Chalmers University of Technology, Gothenburg, Sweden; Curtin University, Perth, WA, Australia; and Aalto University, Espoo, Finland. His research interests include Bayesian estimation, with emphasis on dynamic systems and multiple target tracking.

Dr. Ángel F. García-Fernández was the recipient of paper awards at the International Conference on Information Fusion in 2017, 2019, and 2021.