A Splitting Method for Optimal Control

Brendan O'Donoghue, Giorgos Stathopoulos, and Stephen Boyd

***Abstract*— We apply an operator splitting technique to a generic linear-convex optimal control problem, which results in an algorithm that alternates between solving a quadratic control problem, for which there are efficient methods, and solving a set of single-period optimization problems, which can be done in parallel, and often have analytical solutions. In many cases, the resulting algorithm is division-free (after some off-line pre-computations) and can be implemented in fixed-point arithmetic, for example on a field-programmable gate array (FPGA). We demonstrate the method on several examples from different application areas.**

***Index Terms*— Alternating directions method of multipliers (ADMM), convex optimization, embedded control, fixed point algorithms for control, model predictive control (MPC), operator splitting, optimal control.**

I. Introduction

We consider a linear-convex optimal control problem, i.e., the problem of finding a trajectory of state-control pairs that satisfy a set of linear dynamics, and minimize a sum of convex stage-cost functions. This problem arises naturally in many application areas, including model predictive control (MPC), moving horizon estimation, and trajectory planning. Finding an optimal trajectory involves solving a convex optimization problem, which can, in principle, be done efficiently, using generic techniques for convex optimization, or methods developed specifically for the linear-convex optimal control problem, such as custom interior-point methods. We are interested here in real-time applications, which require very high execution speed and simplicity of the algorithm. On the other hand, most real-time applications do not require the solution to be computed to high accuracy; assuming the variables have been appropriately scaled, three digits of accuracy is usually more than adequate.

In this brief, we describe yet another method to solve linear-convex optimal control problems quickly. The algorithm we present relies on an operator splitting technique, referred to as the alternating direction method of multipliers (ADMM), or as Douglas–Rachford (D-R) splitting. Operator splitting breaks the problem into two parts, a quadratic optimal control problem (which can be solved very efficiently) and a set of single period optimization problems (which can be solved in parallel, often, analytically). An iteration that alternates these two steps then converges to a solution. We demonstrate that our method can solve optimal control problems to an acceptable accuracy very rapidly, indicating that it is suitable for use in, e.g., high-frequency control applications. Another advantage of our method is that in many cases, after some off-line pre-computation, the algorithm requires no division operations. In these cases, it can be implemented in fixed-point arithmetic, for example on a field-programmable gate array (FPGA) for high-speed embedded control.

Manuscript received May 1, 2012; revised September 3, 2012; accepted November 16, 2012. Manuscript received in final form December 1, 2012. Date of publication January 28, 2013; date of current version October 15, 2013. Recommended by Associate Editor C. Lagoa.

B. O'Donoghue and S. Boyd are with the Electrical Engineering Department, Stanford University, Stanford, CA 94305 USA (e-mail: bodono@stanford.edu; boyd@stanford.edu).

G. Stathopoulos is with the Delft Center for Systems and Control Department, Delft University of Technology, Delft 2628, The Netherlands (e-mail: stathopog@gmail.com).

Digital Object Identifier 10.1109/TCST.2012.2231960

II. Convex Optimal Control Problem

A. Problem Description

We consider the (deterministic, discrete-time, finite-horizon) linear-convex optimal control problem

$$\begin{aligned} &\text{minimize} \ \sum_{t=0}^{T} (\phi_t(x_t, u_t) + \psi_t(x_t, u_t)) \\ &\text{subject to} \ x_{t+1} = A_t x_t + B_t u_t + c_t, \quad t = 0, \ldots, T-1 \\ &\qquad\qquad x_0 = x_{\text{init}} \end{aligned} \tag{1}$$

with variables (states) $x_t \in \mathbf{R}^n$ and (controls) $u_t \in \mathbf{R}^m$, $t = 0, \ldots, T$. The stage cost function is split into a convex quadratic part ϕ_t and a convex non-quadratic part ψ_t. The convex quadratic terms $\phi_t : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$ have the form

$$\phi_t(x, u) = (1/2) \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T \begin{bmatrix} Q_t & S_t & q_t \\ S_t^T & R_t & r_t \\ q_t^T & r_t^T & 0 \end{bmatrix} \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}$$

where

$$\begin{bmatrix} Q_t & S_t \\ S_t^T & R_t \end{bmatrix} \succeq 0$$

(i.e., is symmetric positive semidefinite). The non-quadratic terms, $\psi_t : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R} \cup \{\infty\}$, are closed proper convex functions (see, e.g., [1]). Infinite values in the functions ψ_t encode convex constraints on the states and controls. For example, when ψ_t is the indicator function of a closed convex set $\mathcal{C}_t \subset \mathbf{R}^{n+m}$

$$\psi_t(x_t, u_t) = I_{\mathcal{C}_t}(x_t, u_t) = \begin{cases} 0, & (x_t, u_t) \in \mathcal{C}_t \\ \infty, & \text{otherwise} \end{cases}$$

the stage cost term $\psi_t(x_t, u_t)$ in (1) simply imposes the state-control constraint $(x_t, u_t) \in \mathcal{C}_t$.

The problem data are the initial state $x_{\text{init}} \in \mathbf{R}^n$, the dynamics matrices A_t, B_t, and c_t, for $t = 0, \ldots, T-1$, the quadratic cost term coefficients Q_t, R_t, S_t, q_t, and r_t, for $t = 0, \ldots, T$, and the non-quadratic cost functions ψ_t, $t = 0, \ldots, T$. The decomposition of the stage cost into the quadratic part ϕ_t and the non-quadratic part ψ_t is, in general,

not unique, since a quadratic stage cost term can be included in the non-quadratic term.

Problem (1) is a convex optimization problem and as such it can be solved efficiently by a variety of generic methods, including, e.g., interior point methods [2], [3]. Here, "efficiently" means that the computational effort required to obtain a solution grows no faster than a polynomial of the problem size. This brief is about solving the linear-convex optimal control problem (1) very quickly, even by comparison to these generic methods.

B. Usage Scenarios

We list here some ways in which a solver for (1) can be used.

1) Cold Start: Here, we solve (1) just once, or many times, but with very different (unrelated) data each time.

2) Warm Start: Here, we solve (1) many times sequentially, where the data in each problem is similar to the data for the previously solved problem. In this case, we initialize the algorithm with the solution from the previous problem to obtain a speed-up over the cold start scenario; the speed-up will depend on how similar the data are from one iteration to the next.

3) Constant Quadratic Case: Here, we solve (1) many times, where the quantities Q_t, R_t, S_t, $t = 0, \ldots, T$, and A_t, B_t, $t = 0, \ldots, T-1$, do not change. (The initial state and non-quadratic stage cost terms can change in each instance.) In this case, we can cache some quantities that are precomputed, allowing us to reduce the computation required to solve instances of the problem. Depending on the non-quadratic stage cost terms, our algorithm can be division-free; as a result it can be implemented in fixed-point arithmetic on, e.g., an FPGA.

4) Warm Start Constant Quadratic: Here, we have both the warm start and constant quadratic cases, in which case the computational savings stack.

C. Notation

We use $x = (x_0, \ldots, x_T)$ and $u = (u_0, \ldots, u_T)$ to denote the concatenated states and controls (i.e., their trajectories), and $(x, u) \in \mathbf{R}^{(n+m)(T+1)}$ to denote the whole state-control trajectory. We define

$$\mathcal{D} = \{(x, u) \mid x_0 = x_{\text{init}},\ x_{t+1} = A_t x_t + B_t u_t + c_t,\quad t = 0, \ldots, T-1\}$$

which is the set of state-control pairs that satisfy the dynamics of (1), and we use $I_{\mathcal{D}}$ to denote the indicator function of the set $\mathcal{D}$, which is a closed proper convex function. We define

$$\phi(x, u) = \sum_{t=0}^{T} \phi_t(x_t, u_t), \qquad \psi(x, u) = \sum_{t=0}^{T} \psi_t(x_t, u_t)$$

which are the quadratic and non-quadratic costs over the whole trajectory (x, u). With this notation, the convex optimal control problem can be expressed as

$$\text{minimize } I_{\mathcal{D}}(x, u) + \phi(x, u) + \psi(x, u)$$

with variables $(x, u) \in \mathbf{R}^{(n+m)(T+1)}$.

D. Prior and Related Work

In this section, we give a brief overview of some of the important prior work in several related areas.

1) Interior-Point Methods: A generic interior-point solver for (1) that does not exploit the problem structure would scale in complexity with the cube of the time-horizon [4]. If the structure of the problem is exploited, however, the complexity only grows linearly. In [5], the authors developed a custom interior-point method that can solve quadratic optimal control problems with box constraints very rapidly by exploiting problem structure. A similar approach was taken in [6]. For work detailing efficient primal-dual interior-point methods to solve the quadratic programs (QPs) that arise in optimal control, see [7]–[9].

2) Automatic Code Generation: Typically, creating a custom interior-point solver is a very labor-intensive exercise. In [10], the authors described the automatic generation of high speed custom solvers directly from high level descriptions of the problem. These automatically generated custom solvers are tailored to the problem at hand, providing dramatic speed-ups over generic solvers. For work detailing custom code generation specifically for optimal control problems, see [11]–[15].

3) Explicit MPC: Explicit model predictive control is a technique for solving quadratic optimal control problems with polyhedral constraints [16], [17], with all data fixed except the initial state. In this case, the solution is a piecewise affine function of the initial state. The polyhedra that define the regions and the associated coefficients in the affine function, can be computed off-line. Solving the problem then reduces to searching in a lookup table, and then evaluating the affine function (which is division-free). Due to the exponential growth in the number of regions, explicit MPC can realistically only be applied to systems with very modest numbers of states and constraints. For an extension that can handle larger problems by using partial enumeration, see [18].

4) Active Set Methods: Active set methods are a set of techniques for solving QPs that are closely related to the simplex method for linear programming. They rely on identifying the set of constraints that are active at the optimum and then solving a simpler problem using just these constraints [19]–[23]. The use of active set methods to solve the QPs that arise in control has been explored by Ferreau *et al.* in [24] and [25].

5) Fast Gradient Methods: Fast gradient methods, inspired by Nesterov's accelerated first order methods [26], [27], have been applied to the optimal control problem [28]–[32]. These techniques typically require only the evaluation of a gradient and a projection at each iteration. Thus, they generally require less computation than, say, interior-point methods, at the expense of high accuracy.

6) Embedded Control: There has been much recent interest in using MPC in an embedded control setting, for example in autonomous or miniature devices. The challenge is to develop algorithms that can solve (1) quickly, robustly, and within the limitations of on-board chip architectures. Many techniques have been investigated, including interior-point

methods, active set methods, and others; see [33]–[36]. In this brief, we develop an algorithm that (in some cases) can be implemented without division, which allows us to use fixed point arithmetic. For other work exploring the use of fixed point arithmetic in control, see [29], [37], [38]. For other work on implementing control algorithms on FPGAs, see [39]–[43].

7) Operator Splitting: The technique we employ in this brief relies on the work done on monotone operators and operator splitting methods. The history of operator splitting goes back to the 1950s; ADMM itself was introduced in the mid-1970s by Glowinski and Marrocco [44] and Gabay and Mercier [45]. It was shown in [46] that ADMM is a special case of a splitting technique known as Douglas–Rachford splitting, and Eckstein and Bertsekas [47] showed in turn that Douglas–Rachford splitting is a special case of the proximal point algorithm. For convergence, results for operator splitting methods, see [46]–[49]. For (much) more detail about operator splitting algorithms and example applications, see [50] and the references therein. Operator splitting has seen use in many application areas, see [51]–[53]. In [54], the authors used operator splitting to develop sparse feedback gain matrices for linear-quadratic control problems.

Recently, accelerated variants of operator splitting methods have been proposed. Although we do not use these variants in this brief, we mention them here for completeness. These techniques apply a Nesterov-type acceleration to the iterates [26], which under certain conditions can dramatically improve the rate of convergence [55]–[57].

III. Splitting Method

A. Consensus Form

We can write the optimal control problem (1) in the following form:

$$\begin{array}{ll}\text{minimize} & (I_{\mathcal{D}}(x,u)+\phi(x,u))+\psi(\tilde{x},\tilde{u})\\ \text{subject to} & (x,u)=(\tilde{x},\tilde{u})\end{array} \tag{2}$$

with variables $(x,u)\in\mathbf{R}^{(n+m)(T+1)}$, $(\tilde{x},\tilde{u})\in\mathbf{R}^{(n+m)(T+1)}$. This form is referred to as consensus; see [50] and [57]. Here, we split the objective into two separate parts, with different variables. The first term contains the quadratic objective and the dynamic constraints; the second term is separable across time, and encodes the constraints and non-quadratic objective terms on the states and control in each period. The equality constraint requires that they be in consensus.

B. Operator Splitting for Control

The consensus form of operator splitting is the following algorithm. Starting from any initial $(\tilde{x}^0,\tilde{u}^0)$ and (z^0,y^0), for $k=0,1,\ldots$

$$(x^{k+1},u^{k+1}) := \underset{(x,u)}{\operatorname{argmin}}\Big(I_{\mathcal{D}}(x,u)+\phi(x,u)+(\rho/2)\|(x,u)-(\tilde{x}^k,\tilde{u}^k)-(z^k,y^k)\|_2^2\Big) \tag{3}$$

$$(\tilde{x}^{k+1},\tilde{u}^{k+1}) := \underset{(\tilde{x},\tilde{u})}{\operatorname{argmin}}\Big(\psi(\tilde{x},\tilde{u})+(\rho/2)\|(\tilde{x},\tilde{u})-(x^{k+1},u^{k+1})+(z^k,y^k)\|_2^2\Big) \tag{4}$$

$$(z^{k+1},y^{k+1}) := (z^k,y^k)+(\tilde{x}^{k+1},\tilde{u}^{k+1})-(x^{k+1},u^{k+1}). \tag{5}$$

Here, $\rho>0$ is an algorithm parameter, k is the iteration counter, and $(z^k,y^k)\in\mathbf{R}^{(n+m)(T+1)}$ is a (scaled) dual variable associated with the consensus constraint. We shall refer to this technique as operator splitting for control (OSC).

The first step entails minimizing a sum of convex quadratic functions of the state and control, subject to the dynamics, i.e., a convex quadratic control problem. The objective in the second step is separable across time, so the minimization for each time period can be carried out separately, as

$$\left(\tilde{x}_t^{k+1},\tilde{u}_t^{k+1}\right) := \underset{(\tilde{x}_t,\tilde{u}_t)}{\operatorname{argmin}}\Big(\psi_t(\tilde{x}_t,\tilde{u}_t)+(\rho/2)\|(\tilde{x}_t,\tilde{u}_t)-\left(x_t^{k+1},u_t^{k+1}\right)+\left(z_t^k,y_t^k\right)\|_2^2\Big)$$

for $t=0,\ldots,T$. The right-hand side is the proximal (or prox) operator of ψ_t, evaluated at

$$(x_t^{k+1}-z_t^k,u_t^{k+1}-y_t^k).$$

For more details about prox operators and derivations of prox operators for some common functions, see [58], [59].

1) Convergence and Stopping Criteria: Under some mild conditions, the splitting algorithm converges to the solution (assuming there is one); see [50, Sec. III. B]. The primal residual for (2) is given by

$$r^k=(x^k,u^k)-(\tilde{x}^k,\tilde{u}^k)$$

and the dual residual is

$$s^k=\rho((\tilde{x}^k,\tilde{u}^k)-(\tilde{x}^{k-1},\tilde{u}^{k-1})).$$

It can be shown that r^k and s^k converge to zero under OSC. A suitable stopping criterion is when the residuals are smaller than some threshold

$$\|r^k\|_2\le\epsilon^{\text{pri}},\quad \|s^k\|_2\le\epsilon^{\text{dual}}$$

where $\epsilon^{\text{pri}}>0$ and $\epsilon^{\text{dual}}>0$ are tolerances for primal and dual feasibility, respectively. One way to assign these tolerances is the following [50, Sec. III.C]:

$$\begin{aligned}\epsilon^{\text{pri}} &= \epsilon^{\text{abs}}\sqrt{(T+1)(n+m)}\\ &\quad+\epsilon^{\text{rel}}\max\{\|(x^k,u^k)\|_2,\|(\tilde{x}^k,\tilde{u}^k)\|_2\}\\ \epsilon^{\text{dual}} &= \epsilon^{\text{abs}}\sqrt{(T+1)(n+m)}+\epsilon^{\text{rel}}\|(z^k,y^k)\|_2\end{aligned} \tag{6}$$

where $\epsilon^{\text{abs}}>0$ and $\epsilon^{\text{rel}}>0$ are absolute and relative tolerances, respectively. This choice ensures that the primal and dual tolerances scale with the size of the problem and the scale of the typical variable values. In any particular application, however, it can be simpler to choose fixed values for the primal and dual tolerances ϵ^{pri} and ϵ^{dual}.

Douglas–Rachford splitting can take many iterations to converge to high accuracy. However, modest accuracy is typically achieved within a few tens of iterations. This is in contrast to, e.g., interior-point methods, which can converge to high accuracy in a few tens of iterations, but the per iteration cost

is generally much higher. In many practical cases, including much of control, extremely high accuracy is not required and the moderate accuracy provided by the splitting technique is sufficient.

The best known theoretical guarantees of Douglas–Rachford splitting are rather weak: to achieve an error of less than ϵ, D-R splitting requires at most $\mathcal{O}(1/\epsilon^2)$ steps [50, Sec. III]. However, typically, the empirical convergence rate is much faster than this worst-case estimate.

2) Relaxation: In some cases, we can improve the convergence rate of OSC by applying relaxation. Here, we replace (x^{k+1}, u^{k+1}) in (4) and (5) with

$$\alpha(x^{k+1}, u^{k+1}) + (1-\alpha)(\tilde{x}^k, \tilde{u}^k)$$

where $\alpha \in (0, 2)$ is a relaxation parameter; when $\alpha < 1$ this is referred to as under-relaxation, when $\alpha > 1$ it is referred to as over-relaxation. This scheme is analyzed in [47], and experiments in [60] and [61] show that values of $\alpha \in [1.5, 1.8]$ can improve empirical convergence.

C. Quadratic Control Step

The first step in the splitting algorithm can be expressed as a linearly constrained quadratic minimization problem

$$\begin{array}{l}\text{minimize } (1/2)w^T E w + f^T w \\ \text{subject to } Gw = h\end{array}$$

over variable $w \in \mathbf{R}^{(T+1)(n+m)}$, where

$$w = \begin{bmatrix} x_0 \\ u_0 \\ \vdots \\ x_T \\ u_T \end{bmatrix}, \quad f = \begin{bmatrix} q_0 - \rho(\tilde{x}_0^k + z_0^k)) \\ r_0 - \rho(\tilde{u}_0^k + y_0^k) \\ \vdots \\ q_T - \rho(\tilde{x}_T^k + z_T^k) \\ r_T - \rho(\tilde{u}_T^k + y_T^k) \end{bmatrix}, \quad h = \begin{bmatrix} x_{\text{init}} \\ c_0 \\ \vdots \\ c_{T-1} \end{bmatrix}$$

$$E = \begin{bmatrix} Q_0 + \rho I & S_0 & \cdots & 0 & 0 \\ S_0^T & R_0 + \rho I & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & Q_T + \rho I & S_T \\ 0 & 0 & \cdots & S_T^T & R_T + \rho I \end{bmatrix}$$

$$G = \begin{bmatrix} I & 0 & \cdots & 0 & 0 & 0\ 0 \\ -A_0 & -B_0 & \cdots & 0 & 0 & 0\ 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & I & 0 & 0\ 0 \\ 0 & 0 & \cdots & -A_{T-1} & -B_{T-1} & I\ 0 \end{bmatrix}.$$

The matrix E is block diagonal with $T+1$ blocks of size $(n+m) \times (n+m)$; it is positive definite since $\rho > 0$. The matrix G has full (row) rank (i.e., $(T+1)n$), due to the identity blocks.

Necessary and sufficient optimality conditions are (the KKT equations)

$$\begin{bmatrix} E & G^T \\ G & 0 \end{bmatrix} \begin{bmatrix} w \\ \lambda \end{bmatrix} = \begin{bmatrix} -f \\ h \end{bmatrix} \tag{7}$$

where $\lambda \in \mathbf{R}^{(T+1)n}$ are dual variables associated with the equality constraints. This is a set of $(T+1)(2n+m)$ equations in $(T+1)(2n+m)$ variables; the coefficient matrix (which is called the KKT matrix) is invertible since E is positive definite and G is full rank. We must solve (7) multiple times, once per iteration of the splitting algorithm, using the same KKT matrix, but with different values of f.

The traditional approach to solving (7) is to eliminate the variable w using

$$w = -E^{-1}G^T\lambda - E^{-1}f \tag{8}$$

which results in the reduced equation

$$GE^{-1}G^T\lambda = -h - GE^{-1}f$$

for the dual variable λ. The matrix $GE^{-1}G^T$ is block tridiagonal, and can therefore be solved by a Riccati-like recursion in order Tn^3 floating-point operations (flops) [2, Sec. X]. The variable w is then reconstructed from (8). The overall complexity (including forming $GE^{-1}G^T$ and computing z) is order $T(n+m)^3$ flops. Careful analysis of the Riccati algorithm reveals that if we cache or pre-compute various matrices that arise in solving (7), we can solve subsequent instances, with the same KKT matrix but different values of f, in order $T(m+n)^2$ flops.

A modern approach to solving (7) is to use a sparse LDL^T decomposition. We factor the KKT matrix as

$$\begin{bmatrix} E & G^T \\ G & 0 \end{bmatrix} = \text{PLDL}^T P^T$$

where P is a permutation matrix, L is lower triangular with diagonal entries one, and D is block diagonal with 1×1 or 2×2 blocks. The permutation matrix P is chosen based on the sparsity pattern of the KKT matrix, in order to yield a factor L with few nonzeros and a factorization that is stable [62], [63]. We then solve (7) using

$$\begin{bmatrix} w \\ \lambda \end{bmatrix} = P\left(L^{-T}\left(D^{-1}\left(L^{-1}\left(P^T\begin{bmatrix} -f \\ h \end{bmatrix}\right)\right)\right)\right).$$

Multiplication by L^{-1} is carried out by forward substitution, and multiplication by L^{-T} is carried out by backward substitution; these operations do not require division. The forward and backward substitution steps require a number of flops on the order of the number of nonzero entries in L. Inverting D amounts to inverting 2×2 matrices; if we pre-compute the inverse of D, which is block diagonal with the same structure as D, the solve step above requires no division; it relies on addition and multiplication. To solve the KKT system many times, with the same KKT matrix but different values of f, we compute and cache P, L, and D^{-1}; subsequently we only carry out the solve step above.

It can be shown that the Riccati recursion is equivalent to the factor-solve method for a particular choice of P [2, Sec. X-D]. When the blocks in E and G are all dense, the factor-solve method has the same complexity: order $T(m+n)^3$ flops to carry out the initial factorization, and order $T(m+n)^2$ flops to carry out subsequent solves. But the general LDL^T method can allow us to exploit additional sparsity within the matrices to reduce the complexity of both the factor and solve steps.

1) Regularization: To ensure that the factorization always exists and that the factorization algorithm is stable, we can regularize the system (for discussion on the stability of factorization algorithms, see [63]–[65]). Instead of the original KKT matrix, we factor the regularized KKT matrix

$$\begin{bmatrix} E & G^T \\ G & -\epsilon I \end{bmatrix}$$

where $\epsilon > 0$ is a small constant. The regularized matrix is quasi-definite [66], which implies that for any permutation P the factorization exists, with D diagonal, and is numerically stable. It is suggested in [67] that a value as small as $\epsilon = 10^{-8}$ gives stability without seriously altering the problem. It was shown in [47] that the operator splitting method still converges to a solution when the subproblems are not solved exactly, as long as certain conditions on the solutions are met; for example, we can reduce the regularization parameter ϵ as iteration proceeds. The effects of the regularization can also be removed or reduced using iterative refinement [68, Sec. IV], [10]. In practice we find that a fixed and small regularization parameter value works very well, and that iterative refinement is not needed.

D. Single Period Proximal Step

The second step of the splitting algorithm involves solving T problems of the form

$$\text{minimize } \psi_t(x_t, u_t) + (\rho/2)\|(x_t, u_t) - (v_t, w_t)\|_2^2 \qquad (9)$$

over $x_t \in \mathbf{R}^n$ and $u_t \in \mathbf{R}^m$, where $v_t \in \mathbf{R}^n$ and $w_t \in \mathbf{R}^m$ are data. When the non-quadratic cost term ψ_t is separable across the state and control, it splits further into a proximal optimization for the state and one for the control. In the extreme case when ψ_t is fully separable (down to the individual scalar entries), (9) reduces to solving $m + n$ scalar proximal minimization problems.

When ψ_t is a indicator function of a set, the proximal optimization step reduces to a projection of (v_t, w_t) onto the set. For example, suppose the non-quadratic terms ψ_t are used to enforce state and control constraints of the form $\|x_t\|_\infty \leq 1$, $\|u_t\|_\infty \leq 1$, so ψ_t is the indicator function of the unit box (and is separable to the components). Then the proximal minimization has the simple solution

$$x_t = \underset{[-1,1]}{\text{sat}}(v_t), \qquad u_t = \underset{[-1,1]}{\text{sat}}(w_t)$$

where sat is the standard saturation function. The examples below will show other cases where we can solve the single period proximal minimizations analytically.

We can always solve the single period proximal problems using a general numerical method, such as an interior-point method. Assuming the dominant cost is solving for the search step in each interior-point iteration, the cost is order $(n+m)^3$ flops. Therefore, the total cost of finding the proximal step is no more than order $T(n+m)^3$ flops, executed on a single processor; we can reduce this by employing multiple processors and carrying out the proximal optimizations in parallel.

E. Robust Control

Here, we mention briefly how our algorithm generalizes to the robust control case. In robust control we typically have uncertainty in the dynamics, initial state, or both. One common technique to deal with this uncertainty is to generate a set of candidate dynamics equations and initial states and to find a single sequence of control inputs that minimizes the expected costs over these possibilities.

In other words, our system has N possible dynamics equations

$$x_t^{(i)} = A_t^{(i)} x_t^{(i)} + B_t^{(i)} u_t + c_t^{(i)}, \quad t = 0, \ldots, T-1,$$

from initial state $x_0^{(i)}$, for $i = 1, \ldots, N$. We assume that the probability of the system being in state $x_0^{(i)}$ and being subject to the ith set of dynamics equations is given by p_i. These candidates could have been generated, for example, by sampling from a distribution over the parameters and initial states. The goal is to find a single sequence of inputs, u_t for $t = 0, \ldots, T$, that minimizes the expected cost.

This problem fits our problem description if we make the following substitutions: The state is constructed by stacking the candidate states

$$x_t = \begin{bmatrix} x_t^{(1)} \\ x_t^{(2)} \\ \vdots \\ x_t^{(N)} \end{bmatrix}$$

and the dynamics of our stacked system is

$$x_{t+1} = A_t x_t + B_t u_t + c_t$$

where

$$A_t = \begin{bmatrix} A_t^{(1)} & 0 & \cdots & 0 \\ 0 & A_t^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_t^{(N)} \end{bmatrix}, \quad B_t = \begin{bmatrix} B_t^{(1)} \\ B_t^{(2)} \\ \vdots \\ B_t^{(N)} \end{bmatrix}, \quad c_t = \begin{bmatrix} c_t^{(1)} \\ c_t^{(2)} \\ \vdots \\ c_t^{(N)} \end{bmatrix}.$$

The stage cost at time t is written

$$\sum_{i=1}^{N} p_i \left(\phi_t(x_t^{(i)}, u_t) + \psi_t(x_t^{(i)}, u_t) \right)$$

which is a sum of convex quadratics and convex non-quadratic terms and so it matches our framework.

Note that this problem has additional structure that can be exploited. In particular, the KKT matrix in the quadratic control step has significant extra sparsity from the dynamics equations. This will be exploited by the permuted LDL^T algorithm. Similarly for the proximal step: if the ψ_t terms in the objective are separable into terms involving only the state and only the input, then the proximal step problem will decompose over the N state trajectories (in addition to decomposing across time periods), which can then be solved in parallel.

IV. Examples

In this section, we illustrate OSC on four examples, and report on numerical results. For each example we explain the details of the splitting technique, explain how we generated the data for the problem instances, and compare the OSC solution time with CVX [69], [70], a parser-solver that uses SDPT3 [71]. When we report CVX times we report the solve time only, and not the time required to parse and transform the problem. For each example, we present three instances with different problem dimensions.

All computation, with the exception of the results presented in § IV-E, was carried out on a system with a quad-core Intel Xeon processor, with clock speed 3.4 GHz and 16 GB of RAM, running Linux. All examples were implemented in C, and all but one example was implemented in single thread. Note that SDPT3 is able to exploit the structure inherent in the problem and is automatically multithreaded over the four cores using hyperthreads.

When running OSC, we report the time and number of iterations required to reach an accuracy corresponding to ϵ^{abs} and ϵ^{rel} in (6) both set to 10^{-3}. With these tolerances, the objective value obtained by $(\tilde{x}, \tilde{u})$ at termination was never more than 1% suboptimal, as judged by the objective value obtained by CVX. [Note that $(\tilde{x}, \tilde{u})$ at termination may very slightly violate the equality constraints in (1), but are guaranteed to satisfy any constraints embedded in the non-quadratic cost function terms.]

In each case, we give the time required for the factorization of the KKT matrix, cold start solve, and warm start solve. We break down the cost of each iteration into solving the linear quadratic system and solving the single period proximal problems. The results for each example are presented in a table; all reported times are in milliseconds to an accuracy of 0.1 ms. In certain cases the reported time is 0.0 ms, which indicates that the time required for this step was less than 0.05 ms. In the case where we parallelized the single period proximal problems across multiple cores, we mention the single thread and multithread solve times.

When forming the KKT system, we added regularization of $\epsilon = 10^{-6}$ to all examples to ensure existence of the LDL^T factorization. We found that iterative refinement was not needed in any of the examples. To solve the KKT system, we used the sparse LDL and AMD packages written by Tim Davis *et al.* [72]–[74].

For the cold start case, we initialize variables $(\tilde{x}^0, \tilde{u}^0)$ and (z^0, y^0) to zero for all examples and solve the problem. The time required to solve the problem under a cold start can vary by a small amount, so we report the average solve time over 100 runs (the number of iterations is always the same). The worst-case times were never more than 5% more than the average times.

For warm start, we initialize variables $(\tilde{x}^0, \tilde{u}^0)$ and (z^0, y^0) to those returned at the termination of the algorithm with unperturbed data. When warm starting we consider perturbations to the initial state, i.e., we replace x_{init} with some $\hat{x}_{\text{init}}$; this type of perturbation is common in, e.g., MPC. Since no other data are perturbed the matrix does not have to be re-factorized; this corresponds to the warm start constant quadratic usage scenario. For each example, we run 100 warm starts and report average numbers. Across all examples, we found that the worst-case solve time out of the 100 warm starts was no more than 20% longer than the average time.

TABLE I
OSC Results for the Box Constrained Optimal Control Example

	Small	Medium	Large
State dimension n	5	20	50
Input dimension m	2	5	20
Horizon length T	10	20	30
Total variables	77	525	2170
CVX solve time (ms)	400	500	3400
Fast_MPC solve time (ms)	1.5	14.2	2710
Factorization time (ms)	0.1	1.3	16.8
KKT solve time (ms)	0.0	0.1	0.9
Cold start OSC iterations	92	46	68
Cold start OSC solve time (ms)	0.4	4.4	60.5
Warm start OSC iterations	72.6	35.1	39.5
Warm start OSC solve time (ms)	0.3	3.4	35.2

A. Box Constrained Quadratic Optimal Control

We consider a quadratic optimal control problem with box constraints on the input,

$$\begin{array}{ll}\text{minimize} & (1/2)\sum_{t=0}^{T}(x_t^T Q_t x_t + u_t^T R_t u_t) \\ \text{subject to} & x_{t+1} = A_t x_t + B_t u_t, \quad t = 0, \ldots, T-1 \\ & x_0 = x_{\text{init}} \\ & \|u_t\|_\infty \leq 1\end{array}$$

where $Q_t \succeq 0$ and $R_t \succ 0$. We encode the constraints on the action by setting $\psi_t(x_t, u_t) = I_{\|u_t\|_\infty \leq 1}$, so proximal minimization is just saturation.

1) Numerical Instances: For simplicity, we consider the time-invariant case, i.e., $A_t = A$, $B_t = B$, $Q_t = Q$, and $R_t = R$ for all t. The data were all generated randomly; the matrix A was scaled to be marginally stable, i.e., have a largest eigenvalue magnitude of one. The initial state x_{init} was scaled so that at least one input was saturated for at least the first 2/3 of the horizon. For all instances below we chose $\rho = 50$ and relaxation parameter $\alpha = 1.8$. In this example, the time required to take the proximal step was negligible (less than 0.05 ms for the largest instance), as such there was no benefit to parallelization and all computations were performed serially on a single core. For the warm start, we perturbed each entry of the initial vector by a random proportion sampled from a uniform distribution on $[-0.1, 0.1]$.

For this example, we can compare the performance of OSC to FAST_MPC [5]. CVXGEN [10] can solve the smallest problem instance in about 0.1 ms, but the current version of CVXGEN cannot scale to the larger problems. The results are summarized in Table I.

B. Multiperiod Portfolio Optimization

In this example, we consider the problem of managing a portfolio of n assets over a finite time horizon, as described

in [75]. In this case, our state $x_t \in \mathbf{R}^n$ is the dollar amount invested in each asset; the control $u_t \in \mathbf{R}^n$ is the amount of each asset we buy (or sell, when negative) at time t. The dynamics is given by

$$x_{t+1} = \mathrm{diag}(r_t)(x_t + u_t), \quad t = 0, \ldots, T-1,$$

where $r_t > 0$ is the vector of (estimated) returns in period t. This has our form, with $A_t = B_t = \mathrm{diag}(r_t)$ and $c_t = 0$. The stage cost has form

$$\mathbf{1}^T u_t + \kappa_t^T |u_t| + u_t^T \mathrm{diag}(s_t) u_t + \lambda_t (x_t + u_t)^T \Sigma_t (x_t + u_t),$$

where $\kappa_t \geq 0$, $s_t \geq 0$, $\Sigma_t \succ 0$, and $\lambda_t > 0$ are data; the absolute value is elementwise. These terms are, respectively, gross cash in for sales and purchases, a bid-ask spread transaction fee, a quadratic price impact transaction cost, and a quadratic risk penalty. We are subject to the constraints that $x_{\mathrm{init}} = x_T + u_T = 0$, i.e., the trading starts and ends with no holdings, and a long-only constraint, i.e., $x_t + u_t \geq 0$ for all t.

We split the stage cost as

$$\begin{aligned}
\phi_t(x_t, u_t) &= \mathbf{1}^T u_t + u_t^T \,\mathbf{diag}(s_t) u_t + \lambda_t (x_t + u_t)^T \Sigma_t (x_t + u_t) \\
&\qquad t = 0, \ldots, T, \\
\psi_t(x_t, u_t) &= \kappa_t^T |u_t| + I_{x_t + u_t \geq 0}, \qquad t = 0, \ldots, T-1 \quad \text{and} \\
\phi_T(x_T, u_T) &= \kappa_T^T |u_T| + I_{x_T + u_T = 0}.
\end{aligned}$$

In this example, ψ_t is not state-control separable, but it is separable across assets; that is, it is a sum of terms involving $(x_t)_i$ and $(u_t)_i$. To evaluate the proximal operator for $t < T$, we need to solve a set of problems of the form (using the subscript to denote the component, not time period)

$$\begin{aligned}
&\text{minimize } \kappa_i |u|_i + (\rho/2)\left((x_i - v_i)^2 + (u_i - w_i)^2\right) \\
&\text{subject to } x_i + u_i \geq 0,
\end{aligned}$$

with (scalar) variables x_i and u_i. The solution relies on the proximal operator for the absolute value, which is given by soft-thresholding:

$$S_\gamma(z) = \operatorname*{argmin}_y \left(\gamma |y| + (1/2)(y - z)^2\right) = z\left(\frac{1 - \gamma}{|z|}\right)_+.$$

The solution to the above problem is as follows. If $v_i + S_{\kappa_i/\rho}(w_i) \geq 0$ then $x_i = v_i$ and $u_i = S_{\kappa_i/\rho}(w_i)$, otherwise $u_i = S_{\kappa_i/2\rho}((w_i - v_i)/2)$ and $x_i = -u_i$. For $t = T$, the solution is simply $u_i = S_{\kappa_i/2\rho}((w_i - v_i)/2)$ and $x_i = -u_i$.

1) Numerical Instances: For simplicity, we consider the time-invariant case, that is, the data are constant over time. For more details about how the data were generated, see, [75]. The correlations between assets were selected to be between -0.4 and 0.6. The expected returns were chosen so that all the assets made returns from 0.01% to 3% per period. For all instances below we chose $\rho = 0.1$ and relaxation parameter $\alpha = 1.8$. Again in this case, the time required to calculate the proximal step was negligible (less than 0.05 ms for the largest instance) and thus all computation was performed on a single core. For the warm start, we replaced the initial portfolio with a random portfolio sampled from $\mathcal{N}(0, I)$; this produced initial portfolios with a norm of roughly 20% the norm of the maximum position taken in the unperturbed solution. The results are summarized in Table II.

TABLE II
OSC RESULTS FOR THE PORTFOLIO OPTIMIZATION PROBLEM

	Small	Medium	Large
Number of assets n	10	30	50
Horizon length T	30	60	100
Total variables	620	3660	10100
CVX solve time (ms)	800	2100	10750
Factorization time (ms)	0.7	13.3	73.6
KKT solve time (ms)	0.1	0.7	3.2
Cold start OSC iterations	27	41	53
Cold start OSC solve time (ms)	1.5	30.8	177.7
Warm start OSC iterations	5.1	5.9	4.8
Warm start OSC solve time (ms)	0.3	4.4	16.1

C. Robust State Estimation

In this example, we use noisy state measurements to estimate the state sequence for a dynamical system acted on by a process noise. The system evolves according to

$$x_{t+1} = A_t x_t + B_t u_t, \quad t = 0, \ldots, T-1$$

where here u_t is a process noise. Our measurements are

$$y_t = C_t x_t + v_t, \quad t = 0, \ldots, T$$

where $v_t \in \mathbf{R}^p$ is a measurement noise. We assume the process and measurement noises are independent. We know the initial state of the system, x_{init}.

The maximum-likelihood estimate of the state sequence is found by solving the problem

$$\begin{aligned}
&\text{minimize } \sum_{t=0}^{T} h_t(u_t) + g_t(y_t - C_t x_t) \\
&\text{subject to } x_{t+1} = A_t x_t + B_t u_t, \quad t = 0, \ldots, T-1 \\
&\qquad\qquad x_0 = x_{\mathrm{init}}.
\end{aligned}$$

where h_t is the negative log density of u_t, and g_t is the negative log density of v_t. Assuming these are convex functions (i.e., u_t and v_t have log-concave distributions), this problem has our form. The stage cost is state-control separable. If these distributions are Gaussian, this problem is quadratic.

As a more specific example, we take v_t to have a Gaussian distribution, so g_t is convex quadratic, but we take h_t to be the circular Huber function with half-width M

$$h_t(u_t) = \begin{cases} (1/2)\|u_t\|_2^2, & \|u_t\|_2 \leq M \\ M(\|u_t\|_2 - M/2), & \|u_t\|_2 > M. \end{cases}$$

This corresponds to a process noise that is Gaussian for $\|u_t\|_2 \leq M$, with fat (exponential) tails for $\|u_t\|_2 > M$.

The proximal operator for the circular Huber function has the simple expression

$$\begin{aligned}
&\operatorname*{argmin}_u (h_t(u) + (\rho/2)\|u - v\|_2^2) \\
&\qquad = \left(1 - \min\left(\frac{1}{1+\rho}, \frac{M}{\rho\|v\|_2}\right)\right) v.
\end{aligned}$$

TABLE III
OSC RESULTS FOR THE ROBUST STATE ESTIMATION PROBLEM

	Small	Medium	Large
State dimension n	10	30	50
Input dimension m	10	30	50
Measurement dimension p	5	10	20
Horizon length T	30	60	100
Total variables	620	3660	10100
CVX solve time (ms)	700	1900	8000
Factorization time (ms)	0.5	8.7	48.3
KKT solve time (ms)	0.0	0.6	2.5
Cold start OSC iterations	21	25	29
Cold start OSC solve time (ms)	0.9	14.9	76.7
Warm start OSC iterations	7.5	8.0	7.7
Warm start OSC solve time (ms)	0.3	4.8	20.5

1) Numerical Instances: We consider the case where $A_t = A$, $B_t = I$, and $C_t = C$ for all t. All data were generated randomly, and we scaled A so that the system was marginally stable. The measurement noise was sampled from a standard normal distribution, i.e., $v_t \sim \mathcal{N}(0, I)$. The process noise was generated as follows:

$$u_t \sim \begin{cases} \mathcal{N}(0, I) & \text{with probability } 0.75 \\ \mathcal{N}(0, 10I) & \text{with probability } 0.25 \end{cases}$$

so 25% of the noise samples were large outliers. We took the half-width of the Huber function to be $M = 1$. For all instances, we chose $\rho = 0.1$ and relaxation parameter $\alpha = 1.8$. Again the time required to calculate the proximal step was negligible (less than 0.05 ms for the largest instance) and thus all computation was single threaded. For the warm start, we perturbed each entry of the initial vector by a random proportion sampled from a uniform distribution on $[-0.1, 0.1]$. The results are summarized in Table III.

D. Supply Chain Management

We consider a single commodity supply chain management problem. The supply chain network is described by a directed graph, with n nodes corresponding to warehouses or storage locations, and the m edges corresponding to shipment links between warehouses, as well as from sources and to sinks (which supply or consume the good). The state $x_t \in \mathbf{R}^n_+$ is the amount of the commodity stored in each of the n warehouses, and the control $u_t \in \mathbf{R}^m_+$ is the amount of the commodity shipped along each of the m edges. The dynamics is

$$x_{t+1} = x_t + (B^+ - B^-)u_t$$

where $B^+_{ij} = 1$ if edge j enters node i, $B^-_{ij} = 1$ if edge j leaves node i. Each column of the matrix $(B^+ - B^-)$ corresponding to an edge between warehouses has exactly one entry 1 and one entry -1, and all other entries are zero. If the edge connects to a source then that column has 1 and no other entries, if it connects to a sink then that column has just a -1.

The warehouses have a capacity $C \in \mathbf{R}^n_+$, so $0 \le x_t \le C$. We cannot ship out of any warehouse more than is on hand, so we have the constraint $B^- u_t \le x_t$, in addition to $0 \le u_t \le U$, where U gives the maximum possible shipping levels. (For edges that come from sources, we interpret the corresponding entry of U as a maximum possible supply rate; for edges that go to sinks, we interpret the corresponding entry of U as a maximum demand.)

The stage cost consists of quadratic storage charges, linear transportation charges, the linear cost of acquiring the commodity, and the linear revenue we earn from sinks

$$q_t^T x + \tilde{q}_t^T x^2 + r_t^T u_t$$

where $q_t > 0$ and $\tilde{q}_t > 0$ give the storage cost, and the entries of the vector r_t give the transportation cost, (for edges between warehouses), the cost of acquisition (for sources), and the revenue (for sinks).

We split the stage cost as

$$\phi_t(x_t, u_t) = q_t^T x_t + \tilde{q}_t^T x_t^2 + r_t^T u_t$$

and ψ_t the indicator function of the constraints

$$B^- u_t \le x_t \le C, \qquad 0 \le u_t \le U.$$

The proximal step can be found by first noting that the problem can be decomposed across the n nodes, including all outgoing edges from the node. (Edges coming in from sources are readily handled separately, since their only constraint is that they lie in the interval.) For each node, we solve a problem of the form

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^k (u_j - w_j)^2 + (x - v)^2 \\ \text{subject to} \quad & \sum_{j=1}^k u_j \le x \le C \\ & 0 \le u_j \le U_j, \quad j = 1, \ldots, k \end{aligned}$$

over $x \in \mathbf{R}$ and $u \in \mathbf{R}^k$, where k is the out-degree of the node. Introducing a Lagrange multiplier λ for the constraint $\sum_{j=1}^k u_j \le x$, we end up with

$$u_i = \underset{[0,U]}{\text{sat}}\,(w_i - \lambda), \qquad x = \underset{[0,C]}{\text{sat}}\,(v + \lambda).$$

The solution is found as the smallest value of $\lambda \ge 0$ for which $\sum_{j=1}^k u_j \le x$ holds. This value can be found using, for example, bisection.

1) Numerical Instances: To generate the graph, we distributed warehouses randomly on a unit square, and connected warehouses together that were closer than a threshold, selected so that the graph was fully connected. The cost to traverse an edge was proportional to the distance between the warehouses on the square, plus a small perturbation. We created dummy warehouses to act as sources in one corner of the square and others to act as sinks in the opposite corner, and connected them to nearby warehouses. This ensured that to get from a source to a sink required traversing through several intermediary warehouses. The warehouse capacities were all chosen to be two, the edge limits were all chosen to be one. The revenues from sinks were chosen to have a mean of -15 and the cost of acquisition from sources was chosen to have a mean of five.

For all instances, we chose $\rho = 2.5$ and relaxation parameter $\alpha = 1.8$. In this case, the time required to calculate the

TABLE IV
OSC RESULTS FOR THE SUPPLY CHAIN EXAMPLE

	Small	Medium	Large
Warehouses n	10	20	40
Edges m	25	118	380
Horizon length T	20	20	20
Total variables	735	2898	8820
CVX solve time (ms)	500	1200	3300
Factorization time (ms)	0.3	1.3	4.7
KKT solve time (ms)	0.0	0.1	0.3
Single-thread prox step time (ms)	0.1	0.4	1.3
Multithread prox step time (ms)	0.0	0.1	0.4
Cold start OSC iterations	82	77	116
Cold start OSC solve time (ms)	4.6	19.1	88.1
Warm start OSC iterations	21.9	31.0	24.2
Warm start OSC solve time (ms)	1.2	7.5	18.5

TABLE V
EMBEDDED PROCESSOR TIMING RESULTS, COLD START

	Small	Medium	Large
Box constrained quadratic (ms)	8.4	99.9	1750.3
Portfolio optimization (ms)	29.4	879.1	5036.0
Robust state estimation (ms)	17.7	430.3	2244.5
Supply chain management (ms)	125.9	595.9	2817.7

proximal step was not negligible, so we report times for both single-threaded and multithreaded prox step calculation. For the warm start, we perturbed each entry of the initial vector by a random proportion sampled from a uniform distribution on $[-0.1, 0.1]$. The results are summarized in Table IV.

E. Embedded Results

In this section, we present timing results for an embedded processor running the same examples as above. For these experiments, computation was carried out on a Raspberry Pi, which is a credit-card sized, single-board computer costing about $25 and capable of running Linux. The Raspberry Pi has an ARM1176 core with clock speed 1 GHz, 256 MB of RAM, and a floating point unit. It typically draws less than 2 W of power. With the exception of the matrix factorization, all computation was performed in single-precision. Tables V and VI summarize the timing results for the cold start and warm start cases, respectively. The embedded processor requires roughly an order of magnitude more time to solve the problems than the quad-core Intel Xeon machine, with larger problems requiring greater factors. However, even examples with more than 10 000 variables are solved in just a few seconds, and in all cases the embedded processor solved the problem quicker than CVX on the quad-core machine. Typically, embedded applications do not require very high accuracy solutions, and so, in practice, computation times could be reduced by decreasing the termination tolerances.

V. CONCLUSION

In this brief, we demonstrated an operator splitting technique that can solve optimal control problems rapidly and robustly. Small problems were solved in a few milliseconds or less; example problems with on the order of 10 000 variables were solved in tens of milliseconds. The speedup of OSC over other solvers, including fast and custom solvers specifically for the particular problem type, range from tens to thousands. When the dynamics data do not change, the splitting method yields a division-free algorithm, which can be implemented in fixed-point arithmetic.

TABLE VI
EMBEDDED PROCESSOR TIMING RESULTS, WARM START

	Small	Medium	Large
Box constrained quadratic (ms)	5.5	76.4	1019.9
Portfolio optimization (ms)	5.2	129.1	458.8
Robust state estimation (ms)	6.3	138.6	602.5
Supply chain management (ms)	34.0	245.3	592.8

We close with some comments about the applicability of this technique to the case when the stage-cost functions ψ_t are non-convex. This situation is not uncommon in practice. One common example is when a device (say, an internal combustion engine or turbine) can be operated over some range, and in addition switched on or off; an extreme example is where the control u_t is restricted to a finite set of allowed values. In this case, the optimal control problem (1) is non-convex, and generally hard to solve (exactly). For many of these cases, the proximal step can be carried out efficiently (even though the problem that must be solved is non-convex), so the operator splitting technique described in this brief can be applied. In this case, though, we have no reason to believe that the OSC method will converge, let alone to a solution of the problem. On the other hand, when OSC is used on non-convex problems, it is observed in practice that it typically does converge to a good solution. For a more detailed discussion of Douglas–Rachford splitting applied to non-convex problems, see [50, Section IX].

ACKNOWLEDGMENT

The authors would like to thank Y. Wang for helpful discussions and feedback. The authors would also like to thank three anonymous reviewers for their thoughtful and constructive feedback.

REFERENCES

[1] R. T. Rockafellar, *Convex Analysis*. Princeton, NJ: Princeton Univ. Press, 1970.
[2] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, MA: Cambridge Univ. Press, 2004.
[3] S. Wright, "Interior-point method for optimal control of discrete-time systems," *J. Optim. Theory Appl.*, vol. 77, no. 1, pp. 161–187, Apr. 1993.
[4] J. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Philadelphia, PA: SIAM, 2010.
[5] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 2, pp. 267–278, Mar. 2010.
[6] C. Rao, S. Wright, and J. Rawlings, "Application of interior point methods to model predictive control," *J. Optim. Theory Appl.*, vol. 99, no. 3, pp. 723–757, Nov. 2004.
[7] A. Hansson, "A primal-dual interior-point method for robust optimal control of linear discrete-time systems," *IEEE Trans. Autom. Control*, vol. 45, no. 9, pp. 1639–1655, Sep. 2000.

[8] A. Hansson and S. Boyd, "Robust optimal control of linear discrete-time systems using primal-dual interior-point methods," in *Proc. Amer. Control Conf.*, vol. 1. 1998, pp. 183–187.

[9] M. Åkerblad and A. Hansson, "Efficient solution of second order cone program for model predictive control," *Int. J. Control*, vol. 77, no. 1, pp. 55–77, Jan. 2004.

[10] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optim. Eng.*, vol. 13, no. 1, pp. 1–27, 2012.

[11] B. Houska, H. Ferreau, and M. Diehl, "ACADO toolkit-an open-source framework for automatic control and dynamic Opt." *Optim. Control Appl. Methods*, vol. 32, no. 3, pp. 298–312, Jun. 2011.

[12] A. Domahidi, A. Zgraggen, M. Zeilinger, M. Morari, and C. Jones, "Efficient interior point methods for multistage problems arising in receding horizon control," in *Proc. 51st IEEE Conf. Decision Control*, Dec. 2012, pp. 1–7.

[13] T. Ohtsuka and A. Kodama, "Automatic code generation system for nonlinear receding horizon control," *IEEE Trans. Soc. Instrum. Control Eng.*, vol. 38, no. 7, pp. 617–623, Jul. 2002.

[14] T. Ohtsuka, "A continuation/GMRES method for fast computation of nonlinear receding horizon control," *Automatica*, vol. 40, no. 4, pp. 563–574, Apr. 2004.

[15] J. Mattingley, Y. Wang, and S. Boyd, "Receding horizon control: Automatic generation of high-speed solvers," *IEEE Control Syst. Mag.*, vol. 31, no. 3, pp. 52–65, Jun. 2011.

[16] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, Jan. 2002.

[17] P. Tøndel, T. Johansen, and A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit MPC solutions," in *Proc. IEEE Conf. Decision Control*, Dec. 2001, pp. 1199–1204.

[18] G. Pannocchia, J. Rawlings, and S. Wright, "Fast, large-scale model predictive control by partial enumeration," *Automatica*, vol. 43, no. 5, pp. 852–860, May 2006.

[19] M. Best, "An algorithm for the solution of the parametric quadratic programming problem," in *Appl. Math. Parallel Computation*. Physica-Verlag: Heidelberg, 1996, pp. 57–76.

[20] P. Gill, N. Gould, W. Murray, M. Saunders, and M. Wright, "A weighted Gram-Schmidt method for convex quadratic programming," *Math. Programm.*, vol. 30, pp. 176–195, Jan. 1984.

[21] P. Gill, W. Murray, M. Saunders, and M. Wright, "Procedures for optimization problems with a mixture of bounds and general linear constraints," *ACM Trans. Math. Softw.*, vol. 10, no. 3, pp. 282–298, Sep. 1984.

[22] D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Math. Programm.*, vol. 27, no. 1, pp. 1–33, 1983.

[23] R. Bartlett and L. Biegler, "QPSchur: A dual, active set, Schur complement method for large-scale and structured convex quadratic programming algorithm," *Opt. Eng.*, vol. 7, no. 1, pp. 5–32, Mar. 2006.

[24] H. Ferreau, "An online active set strategy for fast solution of parametric quadratic programs with applications to predictive engine control," M.S. thesis, Univ. Heidelberg, Heidelberg, Germany, 2006.

[25] H. Ferreau, H. Bock, and M. Diehl, "An online active set strategy to overcome the limitations of explicit MPC," *Int. J. Robust Nonlinear Control*, vol. 18, no. 8, pp. 816–830, 2008.

[26] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/k^2)$," *Soviet Math. Doklady*, vol. 27, no. 2, pp. 372–376, 1983.

[27] Y. Nesterov. (2007) "Gradient methods for minimizing composite objective function," [Online]. Available: http://www.ecore.be/DPs/dp 1191313936.pdf

[28] S. Richter, C. Jones, and M. Morari, "Real-time input-constrained MPC using fast gradient methods," in *Proc. 48th IEEE Conf. Decision Control*, Dec. 2009, pp. 7387–7393.

[29] S. Richter, S. Marièthoz, and M. Morari, "High-speed online MPC based on a fast gradient method applied to power converter control," in *Proc. Amer. Control Conf.*, Jul. 2010, pp. 4737–4743.

[30] M. Kögel and R. Findeisen, "A fast gradient method for embedded linear predictive control," in *Proc. 18th IFAC World Conf.*, Aug. 2011, pp. 1362–1367.

[31] M. Kögel and R. Findeisen, "Fast predictive control of linear, time-invariant systems using an algorithm based on the fast gradient method and augmented lagrange multipliers," in *Proc. IEEE Int. Conf. Control Appl.*, Sep. 2011, pp. 780–785.

[32] M. Kögel and R. Findeisen, "Fast predictive control of linear systems combining Nesterov's gradient method and the method of multipliers," in *Proc. 50th IEEE Conf. Decision Control*, Dec. 2011, pp. 501–506.

[33] K. Ling, B. Wu, and J. Maciejowski, "Embedded model predictive control (MPC) using a FPGA," in *Proc. 17th IFAC World Congr.*, Jul. 2008, pp. 15250–15255.

[34] M. Lau, S. Yue, K. Ling, and J. Maciejowski, "A comparison of interior point and active set methods for FPGA implementation of model predictive control," in *Proc. ECC*, Aug. 2009, pp. 156–161.

[35] J. Jerez, G. Constantinides, E. Kerrigan, and K. Ling, "Parallel MPC for real-time FPGA-based implementation," in *Proc. 18th IFAC World Congr.*, Aug. 2011, pp. 1338–1343.

[36] S. Longo, E. Kerrigan, K. Ling, and G. Constantinides, "Parallel move blocking model predictive control," in *Proc. IEEE Decision Control Eur. Control Conf.*, Dec. 2011, pp. 1239–1244.

[37] S. Richter, C. Jones, and M. Morari, "Computational complexity certification for real-time MPC with input constraints based on the fast gradient method," *IEEE Trans. Autom. Control*, vol. 57, no. 6, pp. 1391–1403, Jun. 2012.

[38] J. Jerez, G. Constantinides, and E. Kerrigan, "To-wards a fixed point QP solver for predictive control," in *Proc. IEEE 51st Conf. Decision Control*, Dec. 2012, pp. 1–6.

[39] A. Wills, G. Knagge, and B. Ninness, "Fast linear model predictive control via custom integrated circuit architecture," *IEEE Trans. Control Syst. Technol.*, vol. 20, no. 1, pp. 59–71, Jan. 2012.

[40] J. Jerez, G. Constantinides, and E. Kerrigan, "An FPGA implementation of a sparse quadratic programming solver for constrained predictive control," in *Proc. ACM Symp. Field Program. Gate Arrays*, 2011, pp. 209–218.

[41] P. Vouzis, L. Bleris, M. Arnold, and M. Kothare, "A system-on-a-chip implementation for embedded real-time model predictive control," *IEEE Trans. Control Syst. Technol.*, vol. 17, no. 5, pp. 1006–1017, Sep. 2009.

[42] A. Wills, G. Knagge, and B. Ninness, "Fast linear model predictive control via custom integrated circuit architecture," *IEEE Trans. Control Syst. Technol.*, vol. 20, no. 1, pp. 59–71, Jan. 2012.

[43] E. Hartley, J. Jerez, A. Suardi, J. Maciejowski, E. Kerrigan, and G. Constantinides, "Predictive control of a Boeing 747 aircraft using an FPGA," in *Proc. IFAC NMPC Conf.*, Aug. 2012, pp. 1–6.

[44] R. Glowinski and A. Marrocco, "Sur l'approximation, par elements finis d'ordre un, et la resolution, par penalisation-dualité, d'une classe de problems de Dirichlet non lineares," in *Proc. Revue Française d'Automatique, Informatique, et Recherche Opérationelle*, vol. 9. 1975, pp. 41–76.

[45] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite element approximations," *Comput. Math. Appl.*, vol. 2, no. 1, pp. 17–40, 1976.

[46] D. Gabay, "Applications of the method of multipliers to variational inequalities," in *Augmented Lagrangian Methods: Applications to the Solution of Boundary-Value Problems*. M. Fortin and R. Glowinski, Eds. Amsterdam, The Netherlands: North-Holland, 1983.

[47] J. Eckstein and D. Bertsekas, "On the Douglasâ-Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Math. Programm.*, vol. 55, no. 1, pp. 293–318, 1992.

[48] B. He and X. Yuan, *On Non-Ergodic Convergence Rate of Douglas-Rachford Alternating Direction Method of Multipliers*., Jan. 2012, pp. 1–9.

[49] B. He and X. Yuan, "On the $O(1/n)$ convergence rate of the Douglas-Rachford alternating direction method," *SIAM J. Numer. Anal.*, vol. 50, no. 2, pp. 700–709, Mar. 2012.

[50] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.

[51] B. Wahlberg, S. Boyd, M. Annergren, and Y. Wang, "An ADMM algorithm for a class of total variation regularized estimation problems," in *Proc. 16th IFAC Symp. Syst. Identificat.*, Jul. 2012, pp. 1–6.

[52] M. Annergren, A. Hansson, and B. Wahlberg, "An ADMM algorithm for solving ℓ_1 regularized MPC," in *Proc. IEEE 51st Conf. Decision Control*, Mar. 2012, pp. 1–8.

[53] P. Combettes, "The convex feasibility problem in image recovery," in *Proc. Adv. Imag. Electron Phys. Conf.*, vol. 95. 1996, pp. 155–270.

[54] F. Lin, M. Fardad, and M. Jovanovic, "Design of optimal sparse feedback gains via the alternating direction method of multipliers," in *Proc. Amer. Control Conf.*, Jun. 2012, pp. 4765–4770.

[55] T. Goldstein, B. O'Donoghue, and S. Setzer. (2012). *Fast Alternating Direction Optimization Methods* [Online]. Available FTP: ftp://ftp.math.ucla.edu/pub/camreport/cam12-35.pdf

[56] D. Goldfarb, S. Ma, and K. Scheinberg, "Fast alternating linearization methods for minimizing the sum of two convex functions," Cornell Univ. Library, Ithaca, NY, Tech. report, 10-02, 2010.
[57] D. Goldfarb and S. Ma, *Fast Multiple Splitting Algorithms for Convex Optimization* 2011, pp. 1–21,
[58] P. Combettes and J. Pesquet, "Proximal splitting methods in signal processing," in *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, H. Bauschke, R. Burachik, V. E. P. Combettes, D. Luke, and H. Wolkowicz, Eds. New York: SIAM, 2011, pp. 185–212.
[59] L. Vandenberghe. (2012) "Lecture on proximal gradient method," [Online] Available: http://www.ee.ucla.edu/vandenbe/shortcourses/mlss12-convexopt.pdf
[60] J. Eckstein, "Parallel alternating direction multiplier decomposition of convex programs," *J. Opt. Theory Appl.*, vol. 80, no. 1, pp. 39–62, 1994.
[61] J. Eckstein and M. Ferris, "Operator-splitting methods for monotone affine variational inequalities, with a parallel application to optimal control," *INFORMS J. Comput.*, vol. 10, no. 2, pp. 218–235, 1998.
[62] J. Bunch and B. Parlett, "Direct methods for solving symmetric indefinite systems of linear equations," *SIAM J. Numer. Anal.*, vol. 8, no. 4, pp. 639–655, Dec. 1971.
[63] J. Demmel, *Applied Numerical Linear Algebra*. Philadelphia, PA: SIAM, 1997.
[64] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*. Philadelphia, PA: SIAM, 2002.
[65] L. Trefethen and D. Bau, *Numerical Linear Algebra*. Philadelphia, PA: SIAM, 1997.
[66] R. Vanderbei, "Symmetric quasi-definite matrices," *SIAM J. Opti.*, vol. 5, no. 1, pp. 100–113, 1995.
[67] M. A. Saunders, "Cholesky-based methods for sparse least squares: The benefits of regularization," in *Linear and Nonlinear Conjugate Gradient-Related Methods*, L. Adams and J. L. Nazareth, Eds. Philadelphia: SIAM, 1996, pp. 92–100.
[68] I. Duff, A. Erisman, and J. Reid, *Direct Methods for Sparse Matrices*. London, U.K.: Oxford Univ. Press, 1989.
[69] M. Grant, S. Boyd, and Y. Ye. (2006) "CVX: Matlab software for disciplined convex programming," [Online]. Available: https://cvxr.com/cvx/
[70] M. Grant and S. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control* (ser. Lecture Notes in Control and Information Sciences), V. Blondel, S. Boyd, and H. Kimura, Eds. New York: Springer-Verlag, 2008, pp. 95–110.
[71] K. Toh, M. Todd, and R. Tütüncü, "SDPT3: A Matlab software package for semidefinite programming," *Opt. Meth. Softw.*, vol. 11, no. 12, pp. 545–581, 1999.
[72] T. Davis, "Algorithm 849: A concise sparse Cholesky factorization package," *ACM Trans. Math. Softw.*, vol. 31, no. 4, pp. 587–591, Dec. 2005.
[73] P. Amestoy, T. Davis, and I. Duff, "Algorithm 837: AMD, an approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 381–388, 2004.
[74] T. Davis, *Direct Methods for Sparse Linear Systems* (ser. SIAM Fundamentals of Algorithms). Philadelphia, PA: SIAM, 2006.
[75] S. Boyd, M. Mueller, B. O'Donoghue, and Y. Wang. (2012) "Performance bounds and suboptimal policies for multi-period investment," [Online]. Available: http://www.stanford.edu/∼boyd/papers/port_opt bound.html