Receding Horizon Control

AUTOMATIC GENERATION OF HIGH-SPEED SOLVERS

JACOB MATTINGLEY, YANG WANG, and STEPHEN BOYD

Receding horizon control (RHC), also known as *model predictive control* (MPC), [1]–[5] is a feedback control technique that became popular in the 1980s. With RHC, an optimization problem is solved at each time step to determine a plan of action over a fixed time horizon. The first input from this plan is applied to the system. At the next time step we repeat the planning process, solving a new optimization problem with the time horizon shifted one step forward. The optimization problem takes into account estimates of future quantities based on available information at each time step. The control policy involves feedback since real-time measurements are used to determine the control input.

Digital Object Identifier 10.1109/MCS.2011.940571
Date of publication: 16 May 2011

RHC is a nonlinear control policy that handles input constraints, output constraints, and various control objectives. Using RHC, a system can be controlled near its physical limits, often obtaining performance superior to linear control [5]–[7]. RHC applications include a wide range of practical settings, such as industrial and chemical process control [8], supply chain management [9], stochastic control in economics and finance [10], revenue management [11], control of hybrid vehicles [12], automotive applications [13]–[15], and aerospace applications [16].

Although RHC is used for a wide range of applications, it does not always outperform traditional control methods. For some problems, a skilled designer can achieve similar performance by tuning a conventional linear controller, such as a proportional-integral-derivative (PID) controller modified to handle constraints, for example, by

1066-033X/11/$26.00©2011IEEE

saturating control inputs that are outside their limits. These conventional methods are tractable for single-input, single-output systems but become cumbersome for systems with multiple inputs and outputs, especially when taking into account complex objectives, strongly nonlinear dynamics, and constraints.

In RHC, the designer specifies the objective and constraints as part of an optimization problem, whereas in a conventional design process, the designer adjusts controller gains and coefficients to indirectly handle constraints, often by trial and error. While sophisticated RHC tuning procedures can be used to obtain the best performance [5], [17], for many problems little tuning is required since parameters are suggested directly by the application. RHC can also explicitly incorporate additional control information, such as future references and estimates of future disturbances.

One requirement of RHC is that an optimization problem must be solved at each time step. Using conventional numerical optimization techniques, the time taken to solve this problem is often much longer than the time taken to compute the control action for a linear controller. Applications of RHC are thus often limited to systems with sample times measured in seconds, minutes, or hours.

Various techniques can be used to speed up the solution of the optimization problems that arise in RHC. When the numbers of states, inputs, and constraints are small, one approach is *explicit MPC* [18], [19], where a closed-form expression for the solution of the optimization problem, as a function of the current state, is computed offline and stored. The online algorithm performs a lookup table search, followed by control law evaluation, which can be computed quickly. Another method, applicable to a range of problem sizes, is to custom code online optimization solvers that exploit the problem structure that arises in RHC applications [20]–[23]. Although these custom solvers can yield computation times that are several orders of magnitude faster than generic solvers, their development requires time-consuming hand coding as well as significant expertise in optimization algorithms and numerical computation.

In this article, we describe advances that facilitate the development of custom RHC solvers. By combining a high-level specification language for optimization and code-generation tools, a user of RHC can specify and generate fast, reliable custom code. Since the user does not need expertise in optimization, this tool makes RHC accessible to more practitioners. Combined with automatic code generation, RHC offers a framework for the rapid design of sophisticated controllers for a wide range of problems, including those that require kilohertz sample rates.

In the remainder of the article, we give a high-level overview of RHC, briefly explain code generation for RHC using the software package CVXGEN [24], and illustrate the ideas with three examples. The examples are chosen to show the variety of problems that can be addressed and are simplified for pedagogical reasons. See [25] for a more detailed account of convex optimization, [26] for more on parser-solvers and convex programming, and [27] and [28] for a discussion of code generation for convex optimization.

We restrict attention to systems with linear dynamics, convex objective functions, and convex constraints for several reasons. First, many systems can be reasonably modeled in this restricted form. Second, linearization techniques can be used to extend these methods to systems with nonlinear dynamics. For nonlinear control, the system is linearized around predicted state and input trajectories, and a sequence of linear RHC problems is solved to produce a plan of action [29]. This strategy, known as sequential convex optimization [30], is implemented in the software package NEWCON [31]. An example of automatic code generation applied to nonlinear RHC can be found in [32], while application to a two-link robot arm is discussed in [33]. The toolbox ACADO [34] provides a software package for solving RHC problems with nonlinear dynamics.

FORMULATING RHC PROBLEMS

System Dynamics and Control

We consider the discrete-time linear dynamical system

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

where $x_t \in \mathbf{R}^n$ is the system state, $u_t \in \mathbf{R}^m$ is the control input, and $c_t \in \mathbf{R}^n$ is an exogenous input. The matrices $A_t \in \mathbf{R}^{n\times n}$ and $B_t \in \mathbf{R}^{n\times m}$ are the dynamics and input matrices, respectively. The subscripts on A_t, B_t, and c_t indicate that these parameters may change with time.

The state and input must satisfy constraints, which are expressed abstractly as

$$(x_t, u_t) \in \mathcal{C}_t,$$

where $\mathcal{C}_t \subseteq \mathbf{R}^n \times \mathbf{R}^m$ is the constraint set at time t. The instantaneous cost, which is denoted by $\ell_t(x_t, u_t)$, depends on both the current state and control action. The quality of control is judged by the average cost

$$J = \limsup_{N\to\infty} \frac{1}{N} \sum_{t=0}^{N-1} \ell_t(x_t, u_t).$$

If $\ell_t(x_t, u_t)$ is a random variable, we replace $\ell_t(x_t, u_t)$ with its expected value $\mathbf{E}\ell_t(x_t, u_t)$.

The control input u_t is determined using the information available to the controller at time t, including estimates of quantities that are not known, based on information that is known. These estimates are denoted by

$$\hat{A}_{\tau|t}, \quad \hat{B}_{\tau|t}, \quad \hat{c}_{\tau|t}, \quad \hat{\mathcal{C}}_{\tau|t}, \quad \hat{\ell}_{\tau|t}, \quad \hat{x}_{t|t},$$

Combined with automatic code generation, RHC offers a framework for the rapid design of sophisticated controllers for a wide range of problems, including those that require kilohertz sample rates.

where the notation $\hat{z}_{\tau|t}$ denotes an estimate of z_τ based on information available at time t, where $\tau \geq t$. Information available at time t includes conventional information in a control system, such as sensor measurements, as well as known quantities and functions. The available information may also include additional information that is not typically used in a traditional control system, such as historical usage patterns, weather, price trends, and expert forecasts.

These estimates can be obtained in various ways. In the ideal case, we know the quantity being estimated, in which case we replace the estimates with the known value. For example, if the controller has access to perfect measurements of the current state x_t, we take $\hat{x}_{t|t} = x_t$. A traditional method for obtaining estimates is from a statistical model of the uncertain quantities, in which case the estimates are conditional expectations or other statistical estimates, based on the information available at time t. For example, the additive terms c_t are often modeled as independent zero-mean random variables, with the estimate $\hat{c}_{\tau|t} = \mathbf{E}c_t = 0$.

However, the estimates need not be derived from statistical models; for example, future prices can be obtained from a futures market or from analysts who predict trends. Another example arises when the system to be controlled is nonlinear. In this case, $\hat{A}_{\tau|t}$, $\hat{B}_{\tau|t}$, and $\hat{c}_{\tau|t}$ are a linearization of the nonlinear dynamics along a trajectory.

The controller design problem is to find a control policy that chooses the input u_t as a function of the available information at time t, so that the constraints are satisfied and the average cost J is made small.

Receding Horizon Control

The RHC procedure works as follows. At time t, we consider a time interval extending T steps into the future, $t, t+1, \ldots, t+T$. We then carry out the following steps:

1) *Form a predictive model.* Replace all uncertain quantities over the time interval with their current estimates using information available at time t.
2) *Optimize.* The RHC optimization problem takes the form

$$\begin{array}{ll} \text{minimize} & \dfrac{1}{T+1}\sum_{\tau=t}^{t+T}\hat{\ell}_{\tau|t}(\hat{x}_\tau,\hat{u}_\tau) \\ \text{subject to} & \hat{x}_{\tau+1} = \hat{A}_{\tau|t}\hat{x}_\tau + \hat{B}_{\tau|t}\hat{u}_\tau + \hat{c}_{\tau|t}, \quad \tau = t,\ldots,t+T \\ & (\hat{x}_\tau,\hat{u}_\tau) \in \hat{\mathcal{C}}_{\tau|t}, \quad \tau = t,\ldots,t+T \\ & \hat{x}_t = \hat{x}_{t|t}, \end{array} \tag{1}$$

   with variables $\hat{x}_t, \ldots, \hat{x}_{t+T+1}$ and $\hat{u}_t, \ldots, \hat{u}_{t+T}$. The objective in problem (1) is a finite-horizon approximation of the infinite-horizon cost function J. In problem (1), we seek to minimize the estimated objective over the time interval $t, t+1, \ldots, t+T$, subject to the estimated dynamics and constraints. The parameters in this RHC optimization problem are the estimates

$$\hat{A}_{\tau|t}, \quad \hat{B}_{\tau|t}, \quad \hat{c}_{\tau|t}, \quad \hat{\mathcal{C}}_{\tau|t}, \quad \hat{\ell}_{\tau|t},$$

   for $\tau = t, \ldots, t+T$, and the current state estimate $\hat{x}_{t|t}$. The optimal input trajectory of the RHC optimization problem (1), $\hat{u}_t^\star, \ldots, \hat{u}_{t+T}^\star$, is a *plan of action* for the next T steps.
3) *Execute.* We then choose $u_t = \hat{u}_t^\star$ to be the RHC input. At the next time step, the process is repeated, with updated estimates of the current state and future quantities.

We assume that $\hat{\mathcal{C}}_t$ and $\hat{\ell}_t$ are convex, which means that the RHC problem (1) is a convex optimization problem and can be solved using convex optimization tools [25]. Problems with nonconvex objectives and constraints can often be handled using sequential convex optimization, where a sequence of convex problems is solved to find local solutions of the nonconvex problem [30].

In this article, we adopt *certainty-equivalent* RHC, where the uncertain quantities are replaced with estimates. Uncertainty can be dealt with in other ways, for example, by formulating a robust or stochastic RHC optimization problem [5], [7], [35]–[38]. In certainty-equivalent RHC, it is standard practice to add a final state constraint or a final state cost to the RHC problem. In the former case, we add an equality constraint of the form $x_{T+1} = x^{\text{final}}$, or a final constraint set condition $x_{T+1} \in \mathcal{C}^{\text{final}}$. In the latter case, we add $V(x_{T+1})$ to the objective, where V is a cost function for the final state. For some problems, stability guarantees can be obtained by adding specific terminal costs and constraints [1], [3], [5], [7], [39], [40].

Terminal costs and constraints also allow simpler, shorter-horizon controllers to approximate the behavior of controllers with longer horizons. Indeed, we can often obtain near-optimal control performance with horizon $T = 0$ by choosing an appropriate terminal cost function [41]. In this case the control policy, which is also called a control-Lyapunov policy or approximate dynamic programming policy, can be evaluated on time scales measured in tens of microseconds, allowing control at rates exceeding tens of kilohertz [42].

For some applications, constraints on the state variables can lead to an infeasible RHC optimization problem. This problem can occur if an unexpected disturbance affects the system, in which case there may not exist any control policy that keeps the system within the constraints. Infeasibility can also occur if the estimated system parameters are inaccurate so that the RHC optimization problem does not reflect the true system behavior. There are various ways to deal with infeasibility. One strategy is to apply the planned control input from the previous time step. Another method is to allow constraint violations but penalize them in the objective function. These constraints are referred to as *soft constraints* as opposed to the original hard constraints, which cannot be violated [5], [43]. Typically, the constraints that are softened are constraints on the state variables for which some violation may be acceptable. Input constraints, such as actuator limits and trading budgets, usually cannot be violated. Various methods exist for deciding which constraints to relax, based on importance rankings [5]. Although we cannot guarantee that these methods recover feasibility in future time steps, they do allow the controller to compute an acceptable control input when infeasibilities occur.

Except in special cases, the RHC policy is not an optimal policy. In RHC we replace the infinite horizon average cost J with a finite horizon approximation and uncertain quantities with their estimates. We do not solve the controller design problem, which is an infinite-dimensional nonconvex problem, and is usually intractable. Thus, RHC may not achieve the minimum possible average cost among policies that respect the constraints. Instead, RHC is a sophisticated heuristic, as demonstrated on various applications [8]–[16].

DESIGNING AND IMPLEMENTING RHC CONTROLLERS

Designing an RHC Controller

To specify an RHC policy, we must describe three components: a method for estimating uncertain quantities from measurements, including, in particular, the system model; the horizon T; and the terminal costs and constraints. For a given choice of these components, the closed-loop system is simulated offline to assess the performance, and, if needed, the design choices are modified. Simulation of the RHC system requires solving an optimization problem at each time step.

Various software tools [44]–[46] are available for formulating and solving convex problems. These *parser-solvers* take a high-level specification and perform the necessary transformations for solution by a generic convex optimization solver [47]–[49]. Parser-solvers are convenient in design iteration. The user is able to change the objective or constraints in the RHC optimization problem and immediately see the effects in simulation. Parser-solvers are convenient but may lead to slow solve times since each problem instance is parsed and transformed separately. During offline simulation, however, solver speed is not a critical issue; if necessary, simulations can run slower than the required sample rate.

Implementing an RHC Controller

The RHC solver implementation must, of course, run faster than the sample rate. If the solver speed is much faster than the sample rate, we can use a less powerful processor or run the optimization on a processor performing multiple tasks. For applications with a long sample time, measured in seconds or minutes, a parser-solver might be fast enough. However, in many applications a faster solver is required. Even when a parser-solver is fast enough for real-time implementation, we may prefer a simpler solver, involving few or no external libraries, simpler memory management, and a known maximum execution time.

The traditional route to develop such a solver is custom code development, either using a toolbox or from scratch [20], [50]. This process can be labor intensive. In addition, if the problem statement changes, for example, by changing the objective term, all code modifications must be made laboriously, by hand. Small changes in the objective and constraints may significantly change the sparsity pattern of the KKT system, which is a system of linear equations that is solved at every iteration in an optimization algorithm. As a result, minor reformulations may lead to dramatic changes in the code. After that, testing and correcting the code must be done all over again.

Another approach is automatic generation of custom solver code, directly from a high-level description of the problem family. The user describes the problem to be solved in a convenient high-level format; a solver code generator then generates source code for a custom solver of problem instances from the given family. This source code is then compiled, yielding a custom solver. See "Automatic Code Generation" for a detailed comparison of a code generator with a traditional parser-solver.

Automatic code generation yields a solver that is much faster and simpler than a parser-solver, for several reasons. First, algorithm choices, such as the transformation to canonical form or elimination ordering in the linear equation solver, can be made at code-generation time. Second, the generated code can be split into an initialization part, where memory is allocated, and a real-time part, which involves no further memory allocation. The generated code has few branches, which allows the compiler to perform code optimization. Finally, a hard limit on the number of iterations can be imposed, which translates into a known maximum execution time.

In the examples presented in this article we use the automatic code generation software CVXGEN [28], which incorporates all the above features of automatic code generation. CVXGEN handles problems that can be transformed to quadratic programs (QPs). The software generates custom code for transforming the original problem data to the QP format, solving the QP using a

Automatic Code Generation

A conventional parser-solver takes a complete description of a problem instance, including both the problem structure and the problem data. First, the parser-solver transforms the problem data into the standard form for a generic optimization solver. The optimization problem is then solved, and the result transformed back into the original format. Figure S1(a) shows this approach. Parser-solvers facilitate rapid prototyping by allowing the user to change objectives and constraints and re-solve the problem. However, when we solve many problem instances with the same problem structure, the parser-solver carries out these steps for each problem instance.

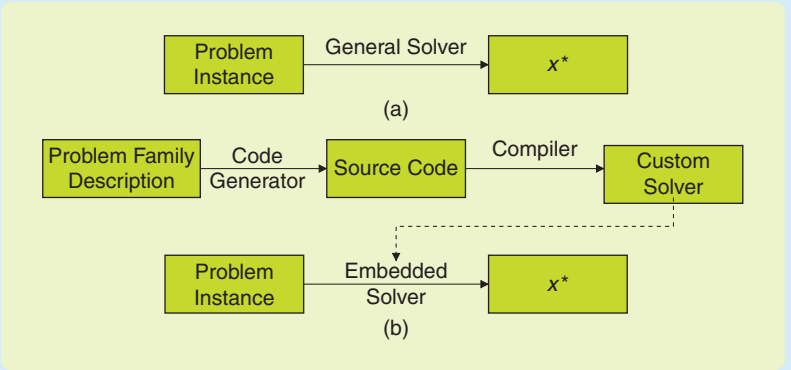

FIGURE S1 (a) shows a general-purpose parser-solver, which takes a single problem instance and outputs the optimal solution. (b) shows a code generator, which takes a problem family description, and produces C code that can be compiled into a customized solver for the problem family. That solver can then be used to solve problem instances.

Instead of performing the same tasks for each problem instance, code generation introduces an additional presolve phase that makes various choices, such as transformations to standard form, entirely offline. The presolve step performs symbolic transformations from the problem family description to an optimization problem in standard form. Once the problem is in standard form, the code generator spends time examining the structure of the problem to determine efficient linear algebra computations, such as factorization orderings, and sparse matrix vector multiplies. The output of code generation is explicit, nearly branch-free C code for transforming the problem into standard form, solving the problem, and transforming the result back. Each step is optimized specifically for the particular problem family, saving significant time when solving each problem instance. The approach is shown in Figure S1(b).

primal-dual interior-point method, and transforming the solution of the QP back to the solution of the original problem. We compare the CVXGEN computation times to the traditional parser-solver CVX [45]. CVX calls either Sedumi or SDPT3 [47], [49], which are generic semidefinite program (SDP) solvers and are not specialized for the specific applications in this article. See "Why CVXGEN Code Is Faster Than Parser-Solvers" for a detailed discussion of why CVXGEN solvers are faster than parser-solvers such as CVX.

EXAMPLES

Preordering

Problem Statement

We consider the problem of meeting a fluctuating demand for a perishable commodity by preordering it with different lead times and also purchasing it on the spot market, all at possibly different prices. When we place an order, we specify delivery for between 1 and n time steps in the future, where faster delivery typically incurs a higher unit cost. Let $u_t \in \mathbf{R}^n_+$ represent new orders, where $(u_t)_i$ is the amount, ordered at time t, to be delivered at time $t+i$. The state is the order book $x_t \in \mathbf{R}^n_+$, where $(x_t)_i$ is the quantity scheduled to arrive at time $t+i-1$; in particular, $(x_t)_1$ is the stock at hand. The system dynamics are $x_{t+1} = Ax_t + Bu_t$, where A is the nilpotent matrix with ones on the upper diagonal and zeros everywhere else, and $B = I$. The constraint has the form $u_t \geq 0$, which is convex. In the model, we take $\mathcal{C}_t = \mathbf{R}^n \times \mathbf{R}^n_+$. Thus, in this example, there is no uncertainty in either the dynamics or constraints.

The stage cost has two terms, namely the cost of placing orders for future delivery, which is incurred when the order is placed, and the cost of making up unmet demand by purchasing on the spot market. The first term has the form $p_t^T u_t$, where $(p_t)_i \geq 0$ is the cost of ordering one unit of the commodity for delivery at time $t+i$. The unmet demand is $(d_t - (x_t)_1)_+$, where $d_t \geq 0$ is the demand at time t, and $(\cdot)_+$ denotes the positive part. The cost of meeting the excess demand on the spot market is $p_t^{\text{spot}}(d_t - (x_t)_1)_+$, where $p_t^{\text{spot}} \geq 0$ is the spot market price at time t. The overall stage cost is thus

$$\ell_t(x_t, u_t) = p_t^T u_t + p_t^{\text{spot}}(d_t - (x_t)_1)_+,$$

which is a convex function of x_t and u_t. Typically, the prices satisfy $p_t^{\text{spot}} > (p_t)_1 > \cdots > (p_t)_n$, which means there is a discount for longer lead time.

Why CVXGEN Code Is Faster Than Parser-Solvers

CVXGEN uses a primal-dual interior-point method, which is an iterative algorithm that computes a sequence of points converging to the optimum. The majority of the computational effort is spent factoring and solving a system of linear equations called the KKT system [25, §10]. Solving an optimization problem requires solving the KKT system around 5–25 times.

One major advantage of code generation is that it allows us to form, factor, and solve the KKT system much faster than with a traditional approach. With code generation we know the problem structure ahead of time, allowing us to move some computational tasks to an offline phase, which is carried out only once. This phase includes determining the required transformations to cast the problem in standard form, determining the required data structures, and choosing an efficient permutation for factoring the KKT system. In contrast, a parser-solver performs these tasks at solve time for each problem instance. Thus, if multiple instances from the same problem family are to be solved, making these choices offline saves considerable time online.

For RHC problems, the KKT system is often sparse [20]. When solving the KKT system, we look for a permutation that yields sparse factor matrices since the number of nonzero entries approximately corresponds to the computational effort of factoring and solving the system. A generic optimization solver cannot spend much time finding an efficient permutation since this task must be done when solving each problem instance. With CVXGEN, the permutation is determined offline, and, therefore, compared with a generic optimization solver, a large amount of time can be used to find an efficient permutation so that the online solve time is as fast as possible.

Another advantage of code generation is in the style of code generated. Since we know the full problem structure offline, code generation allows more explicit code with no dependencies on external linear algebra libraries. For matrix vector multiplication, for example, each entry in the output vector is calculated using a line of C code that refers directly to the relevant entries in the sparse coefficient matrix. This code avoids using sparse matrix routines that unpack, inspect, and repack sparse matrices with each operation, thus reducing computational effort.

For problems with a few tens of variables, CVXGEN produces compact solvers that require less than 100 KB in RAM and ROM combined, which is much less than the memory needed for the bulky external libraries that general purpose solvers use. Thus, for small-sized and medium-sized optimization problems, CVXGEN code is more suited for implementation on an embedded processor, which may have memory restrictions. For larger problems with more than a few thousand variables, CVXGEN produces prohibitively large binary sizes of more than a few megabytes. Code generation can be scaled to larger problems using various methods, but these methods are not yet implemented in CVXGEN. For more details on the implementation see [68].

We consider the case in which the preorder and spot market prices are known and do not vary with time, that is, $p_t = p \in \mathbf{R}^n_+$, $p_t^{\text{spot}} = p^{\text{spot}} \geq 0$, and demand is modeled as a stochastic process. We assume that demand is a stationary log-normal process, which means that $\log d_t$ is a stationary Gaussian process with

$$\mathbf{E} \log d_t = \mu, \qquad \mathbf{E}((\log d_t - \mu)(\log d_{t+\tau} - \mu)) = r_\tau.$$

The mean demand is $\mathbf{E}\, d_t = \exp(\mu + r_0/2)$.

At time t, the controller has access to the current order book x_t, and the current and N most recent values of demand, $d_t, d_{t-1}, \ldots, d_{t-N}$, in addition to the various constants, the prices p and p^{spot}, the log demand mean μ, and the log demand autocovariance r_τ. The orders made at time t are based on this information.

Related Work

An overview of work on supply chain planning without the optimization component is given in [51]. Application of RHC to supply chain problems is discussed in [9] and [52]. In [53], Monte-Carlo simulation of RHC is used to test the sensitivity of various policies, while [54] explores the effects of decentralization. Finally, supply chain optimization with mixed-integer constraints is discussed in [55], while planning under uncertainty is considered in [56].

Receding Horizon Policy

The RHC policy requires estimates of future stage costs, which depend on the unknown future demand. To estimate these costs, we take the exponential of the conditional mean of log demand, given the previous N demand values,

$$\hat{d}_{\tau|t} = \exp \mathbf{E}(\log d_\tau \mid d_t, \ldots, d_{t-N}). \tag{2}$$

Since we know the current demand, we take $\hat{d}_{t|t} = d_t$. The demand is a stationary log-normal process, which means that the conditional expectation of $\log d_\tau$ is an affine function of $\log d_t, \ldots, \log d_{t-N}$. Therefore, it follows from (2)

$$\hat{d}_{\tau|t} = \exp\,(a_{\tau-t}^T(\log d_t, \ldots, \log d_{t-N}) + b),$$

for $\tau = t+1, \ldots, t+T$, where $a_j \in \mathbf{R}^{N+1}$ and $b \in \mathbf{R}$ can be found from the data μ and $r_0, \ldots, r_{N+T+1}$. For this example we also add a terminal constraint $\mathbf{1}^T \hat{x}_{t+T+1} = n\, \mathbf{E}\, d_t$, where

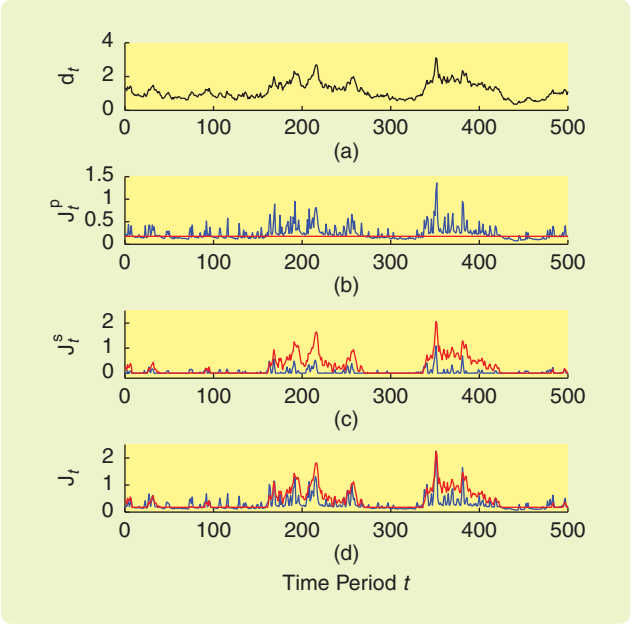

FIGURE 1 Comparison of policies for the preorder example. This figure compares the receding horizon control (RHC) policy (blue) and constant order policy (red) for the preorder example. The plots are (a) demand (d_t), (b) reorder cost ($J_t^{\mathrm{p}} = p^T u_t$), (c) spot market cost ($J_t^{\mathrm{s}} = p^{\mathrm{spot}}(d_t - (x_t)_1)_+$), and (d) stage cost ($J_t = \ell(x_t, u_t)$). The RHC policy performs better overall compared with the constant order policy because it incurs a much smaller spot market cost.

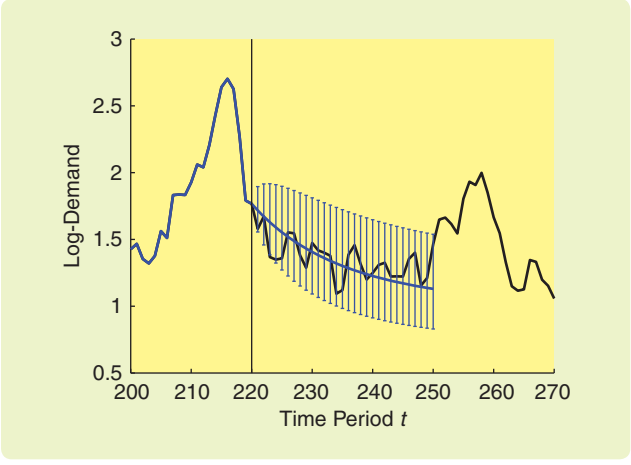

FIGURE 2 Actual and predicted demands for the preorder example. This plot compares actual (black) and predicted (blue) log-demands for the preorder example. The vertical error bars show $\log \hat{d}_{t|220} \pm \sigma_t$. Although the predictions capture the trend, the prediction error can be large.

$\mathbf{E}\, d_t = \exp(\mu + r_0/2)$. This constraint ensures that we do not myopically reduce cost by exhausting inventory at the end of the horizon.

The RHC optimization problem (1) becomes

$$\begin{aligned} \text{minimize} \quad & \frac{1}{T+1}\sum_{\tau=t}^{t+T} p^T \hat{u}_\tau + p^{\mathrm{spot}}(\hat{d}_{\tau|t} - (\hat{x}_\tau)_1)_+ \\ \text{subject to} \quad & \hat{x}_{\tau+1} = A\hat{x}_\tau + \hat{u}_\tau, \quad \tau = t, \ldots, t+T, \\ & \hat{u}_\tau \geq 0, \quad \tau = t, \ldots, t+T, \\ & \mathbf{1}^T \hat{x}_{t+T+1} = n\, \mathbf{E}\, d_t, \quad \hat{x}_t = x_t, \end{aligned}$$

with variables $\hat{x}_t, \ldots, \hat{x}_{t+T+1}$ and $\hat{u}_t, \ldots, \hat{u}_{t+T}$. This optimization problem is convex and can be reduced to an LP.

Constant-Order Policy

We compare the RHC policy with the following policy. At each time t, we let $u_t = (0, \ldots, 0, \overline{u})$. Assuming that the maximum lead-time corresponds to the lowest price, we let $\overline{u} = \mathbf{E}\, d_t = \exp(\mu + r_0/2)$. Thus, in this policy we order with maximum lead-time, an amount equal to the average demand.

Numerical Example

We consider an example with $n = 5$ order lead times and prices

$$p_{\mathrm{spot}} = 1, \qquad p = (\gamma, \gamma^2, \gamma^3, \gamma^4, \gamma^5),$$

where $\gamma = 0.7$. Thus, we have a constant 30% discount for each step of lead time. The demand process data are $\mu = 0$ and $r_\tau = 0.1(0.95^\tau)$. The RHC controller uses horizon $T = 30$, and we estimate future demand using the $N = 100$ most recent demands.

Results

We simulate the RHC and constant-order policies for 1000 steps with the same demand realization. The constant-order policy incurs an average cost $J = 0.37$, while the RHC policy performs better, with an average cost $J = 0.28$. Some example trajectories are shown in Figure 1. We compare the costs incurred by the RHC policy (blue) and constant-order policy (red), over 500 time steps. The plots show demand, preorder cost and spot market costs, and overall stage cost. In Figure 2 we show the log-demand trajectories for a selected time region. The vertical lines show $\exp(\log \hat{d}_{t|220} \pm \sigma_t)$, where $\sigma_t = (\mathbf{E}(\log d_t - \log \hat{d}_{t|220}))^{1/2}$. We see that although the predicted trajectory captures the trend, there is a large prediction error.

The CVXGEN code, shown in Figure 3, requires up to 250 μs to solve at each time step, which is 4000× faster than CVX. When this speed is much faster than needed for a particular application, the extra speed is useful for testing different scenarios and ordering strategies, which require Monte Carlo simulation. Computational performance details are collected in Table 1.

Processor Speed Control

Problem Statement

In this example, a single processor handles jobs from a set of n queues. At each time step the processor adjusts the work rate for each queue. The total work rate determines the processor clock speed, which in turn determines the

power dissipated by the processor. The goal is to adjust the rates to optimally balance average processor power dissipation and queue length.

We use a discrete-time formulation, with state $x_t \in \mathbf{R}^n_+$ and input $u_t \in \mathbf{R}^n_+$, where $(x_t)_i$ is the amount of work to be done in queue i, and $(u_t)_i$ is the work rate for queue i, at time t. The dynamics are $x_{t+1} = x_t - u_t + a_t$, where $a_t \in \mathbf{R}^n_+$ denotes the new work arriving in each queue between times t and $t+1$. At each time we cannot process more than the available work in each queue, which means that $u_t \preceq x_t$. The total work rate of the processor over all queues is $\mathbf{1}^T u_t$.

The processor speed at time t is a function of the work rate vector u_t, namely,

$$s_t = \max\{S^{\min}, \mathbf{1}^T u_t\},$$

where $S^{\min}$ is the minimum allowed processor speed. The processor has the maximum allowed processor speed, $S^{\max}$, which translates to the constraint $\mathbf{1}^T u_t \le S^{\max}$. The processor power dissipation is modeled as αs_t^2, where $\alpha > 0$.

With each queue we associate a linear-plus-quadratic cost $c_i(x_t)_i + d_i(x_t)_i^2$, where c_i and d_i are positive weights. We can interpret c_i as relative queue priorities, when the queues are small, and c_i/d_i as the queue length at which the cost is twice the linear cost alone. When the queue lengths are random variables, the expected value of the queue cost is c_i times the mean queue length, plus d_i times the mean-square queue length. The overall stage cost is

$$\ell(x_t, u_t) = \alpha \max\{S^{\min}, \mathbf{1}^T u_t\}^2 + c^T x_t + d^T x_t^2,$$

where x_t^2 is interpreted componentwise.

For this example the dynamics matrices, constraints, and stage costs are assumed to be known. The only uncertainty is the arrivals a_t, which we assume has the form

$$(a_t)_i = \exp(\lambda_i \sin(2\pi t/M - \theta_i) + (w_t)_i), \quad i = 1, \ldots, n,$$

where M is the period, λ_i, θ_i are known constants, and w_t is a white Gaussian process with mean μ and covariance Σ. At time t, the controller chooses the work rates u_t based on knowledge of the current state x_t, as well as the data $S^{\min}$, $S^{\max}$, α, a, b, λ, θ, μ, Σ, and M.

Related Work

Overviews of power-aware processor design are given in the survey papers [57]–[59]. Closely related work appears in [60], which uses a dynamic speed scaling scheme, motivated by queueing theory, to balance energy consumption and mean response time in a multiprocessor system. In [60], the problem is formulated as a stochastic dynamic program, with an upper bound used instead of an exact solution. In [61], a related problem considered, where the goal is to maximize processing speed while respecting system temperature limits.

```
dimensions
  T = 30; n = 5
end

parameters
  A (n,n); p (n,1)
  d[t], t=0..T
  pspot positive; ubar
  x[0] (n)
end

variables
  x[t] (n), t=1..T+1
  u[t] (n), t=0..T
end

minimize
  (1/(T+1))*sum[t=0..T](p'*u[t]
   + pspot*pos(d[t] - x[t][1]))
subject to
  x[t+1] == A*x[t] + u[t], t=0..T
  u[t] >= 0, t=0..T
  sum(x[T+1]) == n*ubar
end
```

FIGURE 3 CVXGEN code segment for the preorder example. The problem is formulated in a high-level specification language. CVXGEN parses this description and generates code for a solver function, which accepts the declared parameters as input arguments, and solves the optimization problem. Parameters can be declared with attributes. For example, in the above code segment the parameter `pspot` is declared with the attribute `positive`, which is required to ensure that the objective function is convex.

TABLE 1 CVXGEN computational performance comparisons. This table summarizes the performance of the automatically generated code for each example. In all cases, the solve times are on the order of milliseconds, which means a receding horizon controller can be implemented at kilohertz sampling rates. Although Computer 1 is around a factor of ten slower than Computer 3, it uses far less power, namely, 2 W instead of 95 W, as shown in Table 2. We also see that CVXGEN-customized code is more than 1000× faster than the generic solvers Sedumi or SDPT3.

	Preorder	Storage	Processor
CVX (ms)	970	1290	4190
Variables, original	310	153	112
Variables, transformed	341	153	279
Constraints, transformed	373	357	465
KKT matrix nonzeros	1116	1121	1960
KKT factor fill-in	1.64	1.45	1.65
Max steps required	10	16	19
CVXGEN, Computer 1 (ms)	2.34	4.01	7.80
CVXGEN, Computer 2 (ms)	0.96	1.98	3.64
CVXGEN, Computer 3 (ms)	0.25	0.36	0.85

Receding Horizon Policy

In the RHC policy, we take estimates of the arrivals to be

$$(\hat{a}_{\tau|t})_i = \mathbf{E}(a_t)_i = \exp(\lambda_i \sin(2\pi t/M - \theta_i) + \mu_i + 0.5\Sigma_{ii}),$$
$$i = 1, \ldots, n, \quad \tau = t, \ldots, t+T.$$

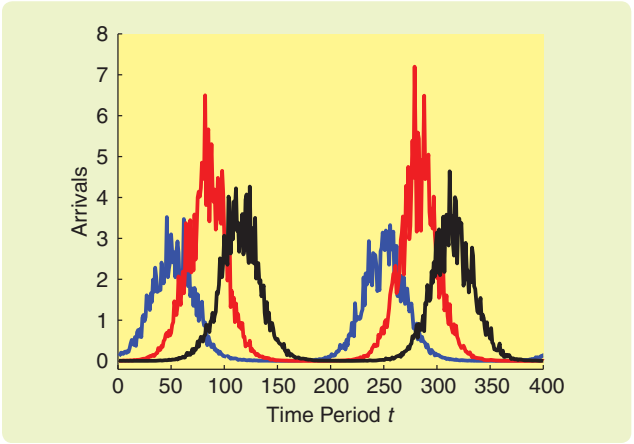

FIGURE 4 Job arrivals for processor speed control. The sample trajectories correspond to $(a_t)_1$ (blue), $(a_t)_2$ (red), and $(a_t)_3$ (black). Note that the arrivals in different queues have different magnitudes and arrival times.

The RHC optimization problem becomes

$$\begin{aligned}\text{minimize}\quad & \frac{1}{T+1}\sum_{\tau=t}^{t+T}\alpha\max\{S^{\min},\mathbf{1}^T\hat{u}_\tau\}^2+c^T\hat{x}_t+d^T\hat{x}_t^2\\ \text{subject to}\quad & \hat{x}_{\tau+1}=\hat{x}_\tau-\hat{u}_\tau+\hat{a}_{\tau|t},\quad \tau=t,\ \ldots,t+T,\\ & 0\le\hat{u}_\tau\le\hat{x}_\tau,\quad \mathbf{1}^T\hat{u}_\tau\le S^{\max},\quad \tau=t,\ \ldots,t+T,\\ & \hat{x}_t=x_t,\end{aligned}\tag{3}$$

where the variables are $\hat{x}_t,\ldots,\hat{x}_{t+T+1}$ and $\hat{u}_t,\ldots,\hat{u}_{t+T}$. This optimization problem can be transformed into a QP.

Proportional Policy

A simple policy is to set the work rates to be proportional to the amount of work left in each queue. Specifically, we take

$$(u_t)_i=\min\{(x_t)_i,\ ((x_t)_i/\mathbf{1}^Tx_t)S^{\max}\}.$$

Taking the minimum ensures that $u_t\le x_t$ is satisfied. The constraint $s_t\le S^{\max}$ is also satisfied by this policy.

Numerical Instance

We consider a numerical example with $n=3$ queues and problem data

$$S^{\min}=1,\quad S^{\max}=5,\quad \alpha=2,\quad c=(1,1,1),\quad d=(0.5,0.5,0.5),$$

and

$$\lambda=(3,3.5,3.2),\quad \theta=(0,1,2),\quad \mu=(-2,-2,-2),$$
$$\Sigma=\mathrm{diag}((0.04,0.04,0.04)).$$

Typical arrival trajectories are shown in Figure 4. For the RHC policy we use the horizon $T=30$.

Results

We simulate both policies for 1000 time steps with the same arrival realization. The RHC policy incurs an average cost of $J=71.3$, while the proportional policy achieves $J=95.3$, which is around 34% worse. Figure 6 shows some sample trajectories. We compare the RHC policy with the proportional policy.

```
dimensions
  n = 3; T = 30
end

parameters
  ahat[t] (n), t=0..T
  alpha positive
  lambda positive
  c (n)
  D (n,n) psd diagonal
  x[0] (n)
  Smin positive
  Smax positive
end

variables
  x[t] (n), t=1..T+1
  u[t] (n), t=0..T
end

minimize
  alpha*sum[t=0..T](sum(square( pos(max(Smin,
  sum(u[t])))) )))
   + sum[t=0..T+1](c'*x[t]) + sum[t=0..T+1]
     (quad(x[t], D))
   + lambda*sum[t=0..T](sum(square(u[t])))
subject to
  x[t+1] == x[t] - u[t] + ahat[t], t=0..T
  u[t] >= 0, t=0..T
  u[t] <= x[t], t=0..T
  sum(u[t]) <= Smax, t=0..T
end
```

FIGURE 5 CVXGEN code segment for processor speed control. The problem is specified in a high-level language in a form that resembles the mathematical problem statement. This approach is convenient for debugging the receding horizon control policy and making changes to the objective and constraints.

The CVXGEN code, shown in Figure 5, takes at most 0.9 ms to solve at each time step, which is 5000× faster than with CVX. Thus, a processor can adjust work rates at 1 kHz using RHC. Alternatively, the same processor can use 1% of its processing power to adjust its own rates at 10 Hz.

Energy Storage

Problem Statement

We consider an energy storage system that can be charged or discharged from a source with varying energy price. An example is a battery connected to a power grid. The goal is to alternate between charging and discharging to maximize the average revenue.

Let $q_t\ge 0$ denote the charge in the energy store at time step t. The energy store has capacity C, and thus $q_t\le C$. The amount of energy taken from the source at time t to charge the energy store is denoted by $u_t^c\ge 0$, and the amount of

Receding horizon control offers a straightforward method for designing feedback controllers that deliver good performance while respecting complex constraints.

energy discharged into the source from the energy store is denoted by $u_t^d \geq 0$. For the problem we consider, at most one of these is positive; that is, we never charge and discharge the store simultaneously. The charging and discharging rates must satisfy

$$u_t^c \leq C^{\max}, \quad u_t^d \leq D^{\max},$$

where $C^{\max}$ and $D^{\max}$ are the maximum charge and discharge rates.

Charging increases the energy in the store by $\kappa^c u_t^c$, where $\kappa^c \in (0, 1)$ is the charge efficiency; discharging decreases the energy in the store by u_t^d/κ^d, where $\kappa^d \in (0, 1)$ is the discharge efficiency. In each time step the energy store leaks, losing energy proportional to its charge, with leakage coefficient $\eta \in (0, 1)$. Incorporating all these effects, the system dynamics are

$$q_{t+1} = \eta q_t + \kappa^c u_t^c - u_t^d/\kappa^d.$$

In the context of the framework for this article, the dynamics matrices are $A = \eta$ and $B = (\kappa^c, 1/\kappa^d)^T$, with $u_t = (u_t^c, u_t^d)$.

The revenue at time t is given by $p_t(u_t^d - u_t^c)$, where p_t is the energy price at time t. To discourage excessive charging and discharging, a penalty of the form $\gamma(u_t^c + u_t^d)$ is added, where $\gamma \geq 0$ is a parameter. An alternative interpretation of this term is a transaction cost, with bid-ask spread γ. Energy is purchased at price $p_t + \gamma$, and sold back at price $p_t - \gamma$. The stage cost is

$$\ell_t(q_t, u_t) = p_t(u_t^c - u_t^d) + \gamma(u_t^c + u_t^d) = (p_t + \gamma)u_t^c - (p_t - \gamma)u_t^d,$$

which can be interpreted as the negative profit, at time t.

We model the energy price as a stationary log-normal process with

$$\mathbf{E}\log p_t = \mu, \quad \mathbf{E}(\log p_t - \mu)(\log p_{t+\tau} - \mu) = r_\tau.$$

At time step t the controller has access to the current charge level q_t, the data C, $C^{\max}$, $D^{\max}$, κ^c, κ^d, η, γ, the current and N most recent prices $p_t, p_{t-1}, \ldots, p_{t-N}$, as well as the mean and autocovariance μ and r_τ. The future prices are not known.

Related Work

A distributed energy system where individual grid-connected households use an MPC-based controller to control micro combined heat and power plants is considered in [62]. For more on distributed generation and variable pricing, see, respectively, [63] and [64]. On the generation side, [65] applies MPC in a case study to wind turbines with batteries to smooth the power produced. A related application is to hybrid vehicles, where MPC-based approaches are developed in [66] or [67]. A vehicle with multiple energy storage units is considered in [12].

Receding Horizon Policy

To implement the receding horizon policy, we take the estimates of future prices to be

$$\hat{p}_{\tau|t} = \exp\mathbf{E}(\log p_\tau \mid p_t, \ldots, p_{t-N}), \quad \tau = t+1, \ldots, t+T.$$

Note that $\hat{p}_{\tau|t}$ is not the same as $\mathbf{E}(p_\tau \mid p_t, \ldots, p_{t-N})$, which can also be computed and used as an estimate. The estimates of future stage costs are

$$\hat{\ell}_t(\hat{q}_\tau, \hat{u}_\tau) = (\hat{p}_{\tau|t} + \gamma)\hat{u}_\tau^c - (\hat{p}_{\tau|t} - \gamma)\hat{u}_\tau^d.$$

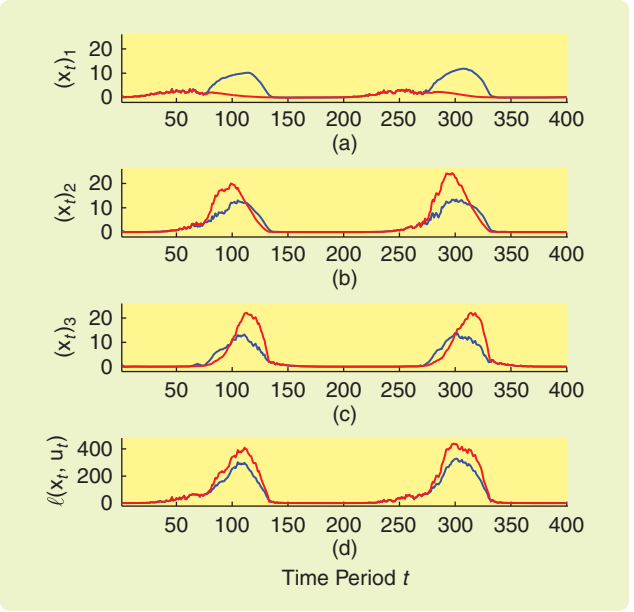

FIGURE 6 Comparison of policies for the processor speed control example. The plots are (a) queue lengths $(x_t)_1$, (b) $(x_t)_2$, (c) $(x_t)_3$, and (d) stage cost $\ell(x_t, u_t)$ for the receding horizon control (RHC) policy (blue) and the proportional policy (red). The proportional policy performs better for queue 1; however, the RHC policy performs better than the proportional policy for queues 2 and 3, and achieves a lower cost overall.

Receding horizon control combined with automatic code generation is a framework for designing and implementing feedback controllers.

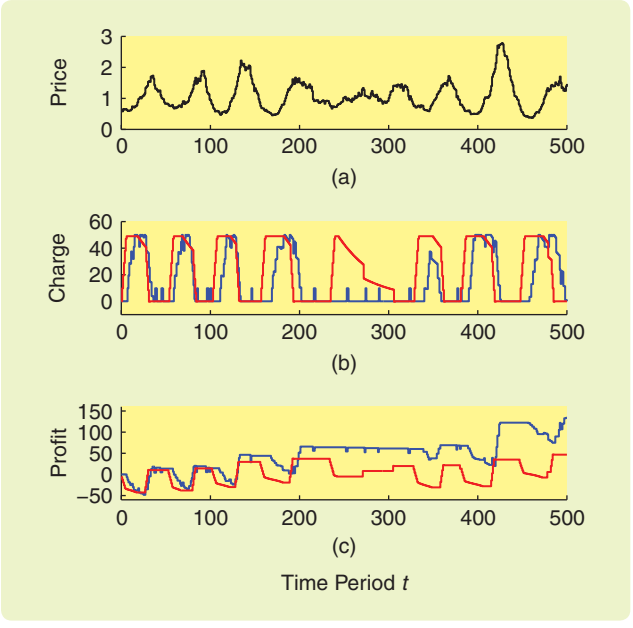

FIGURE 7 Comparison of policies for energy storage. This figure compares the receding horizon control (RHC) policy (blue) and the thresholding policy (red) for the storage example. The plots are (a) price (p_t), (b) charge (q_t), and (c) cumulative profit (r_t). By charging and discharging at the right times, we see that RHC achieves a greater cumulative profit than the thresholding policy.

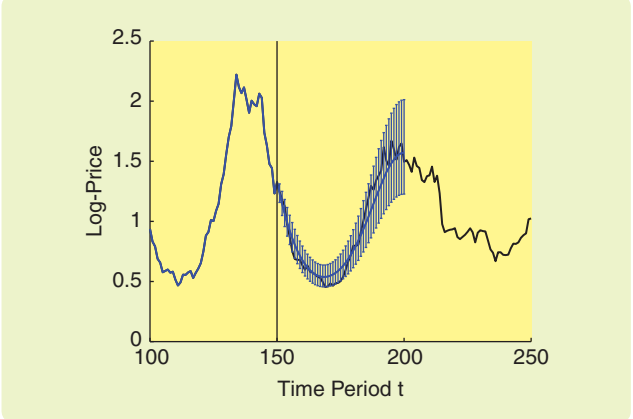

FIGURE 8 Actual and predicted prices for energy storage. Here we compare actual (black) and predicted (blue) log-prices for the storage example. The vertical error bars show $\log \hat{p}_{t|150} \pm \sigma_t$. As with the preorder example, the predictions capture the price trend, although the prediction error increases toward the end of the horizon.

Thus, the RHC optimization problem becomes

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{\tau=t}^{t+T} \hat{\ell}_t(\hat{q}_\tau, \hat{u}_t) \\ \text{subject to} \quad & \hat{q}_{\tau+1} = \eta \hat{q}_\tau + \kappa^c \hat{u}_\tau^c - \hat{u}_\tau^d / \kappa^d, \\ & 0 \le \hat{u}_\tau^c \le C^{\max}, \;\; 0 \le \hat{u}_\tau^d \le D^{\max}, \;\; \tau = t, \ldots, t+T, \\ & 0 \le \hat{q}_\tau \le C, \;\; \tau = t, \ldots, t+T+1, \\ & \hat{q}_\tau = q_t, \end{aligned} \tag{4}$$

with variables $\hat{q}_t, \ldots, \hat{q}_{t+T+1}$, $\hat{u}_t^c, \ldots, \hat{u}_{t+T}^c$, and $\hat{u}_t^d, \ldots, \hat{u}_{t+T}^d$. This optimization problem can be transformed into an LP.

Thresholding Policy

We compare the receding horizon policy with a thresholding policy, which takes

$$u_t^c = \begin{cases} \min(C^{\max}, C - q) & p_t \le p_{\text{thc}} \\ 0 & \text{otherwise,} \end{cases}$$

$$u_t^d = \begin{cases} \min(D^{\max}, q) & p_t \ge p_{\text{thd}} \\ 0 & \text{otherwise.} \end{cases}$$

In other words, the policy charges at the maximum rate if the price is below the threshold p_{thc}, and discharges at the maximum rate if the price is above the threshold p_{thd}. If the price is between p_{thc} and p_{thd} we do not charge or discharge. The minimum is taken to ensure that the charge and discharge constraints are satisfied.

Numerical Example

Let $\eta = 0.98$, $\kappa^c = 0.98$, $\kappa^d = 0.98$, $C^{\max} = 10$, $D^{\max} = 10$, $C = 50$, $\gamma = 0.02$, $q_0 = 0$, $\mu = 0$, and $r_\tau = 0.1(0.99^\tau \cos(0.1\tau))$. For the receding horizon policy we use a time horizon of $T = 50$ steps along with $N = 100$ previous prices to estimate future prices.

Results

The simulations are carried out for 1000 time steps. Figure 7 shows the cumulative profit

$$r_t = \sum_{\tau=0}^{t} p_\tau (u_\tau^d - u_\tau^c) - \gamma (u_\tau^d + u_\tau^c),$$

for the RHC policy (blue) and the thresholding policy (red), over 500 time steps. For the thresholding policy, we adjust the charge and discharge thresholds using trial and error to achieve good performance. The final thresholds, after

trial and error, are $p_{\text{thc}} = 0.8$ and $p_{\text{thd}} = 1.3$. We see that the RHC policy outperforms the thresholding policy. The average profit achieved for the RHC policy is 0.23 per step, whereas thresholding achieves a profit of 0.029 per step, averaged over 1000 time steps.

Figure 8 shows the actual (black) and predicted (blue) log-price trajectories starting at $t = 150$. The vertical lines show $\exp(\log \hat{p}_{t|150} \pm \sigma_t)$, where $\sigma_t = (\mathbf{E}(\log p_t - \log \hat{p}_{t|150}))^{1/2}$. The CVXGEN code, shown in Figure 9, takes at most 360 μ s to solve at each time step, which is 3500× faster than CVX. Further computational performance details are collected in Table 1.

CVXGEN PERFORMANCE

To give an overview of the performance of CVXGEN, we test the code generated by CVXGEN for each example on three different computers. Several extensions can further improve performance, often reducing speed by an order of magnitude or more. First, single-precision floats can be used in place of double precision since the scale of data is known ahead of time. Second, the time horizon selected for the examples is long. With a suitable choice of final state cost, the horizon can be reduced, giving a speedup proportional to the horizon [41]. Finally, the problems are solved to high numerical precision, which requires up to 15–20 steps. With a small amount of tuning, adequate control performance is achievable using a fixed step limit of around five steps [20]. Thus, all of the numerical results are only preliminary upper bounds on performance. The computer properties are summarized in Table 2. We use `gcc-4.4` on each processor, with the compiler optimization flag `-Os`.

In each case, we ensure the computer is idle and then solve the optimization problem instances continuously for at least 1 s. We calculate the maximum time taken to solve an instance, ensuring that each problem is solved to sufficient accuracy so that control performance is not affected.

To compare these computation times to a traditional parser-solver, we also test the performance of CVX on the fastest computer, Computer 3, using Matlab 7.9 and CVX 1.2. For `preorder` and `storage` we set the solver used by CVX to Sedumi 1.2; for `proc_speed` we select SDPT3 4.0. A comparison of the computation times is shown in Table 1. The times listed for CVX are actual solver times and do not include the time required to transform the problem into standard form.

```
dimensions
  T = 50
end

parameters
  eta; kappac; kappad; Cmax; Dmax
  gamma; C; p[t], t=0..T; q[0]
end

variables
  q[t], t=1..T+1
  uc[t], t=0..T
  ud[t], t=0..T
end

minimize
  sum[t=0..T]((p[t] + gamma)*uc[t] - (p[t]
   - gamma)*ud[t])
subject to
  q[t+1] == eta*q[t] + kappac*uc[t]
   - (1/kappad)*ud[t], t=0..T
  0 <= q[t] <= C, t=1..T+1
  0 <= uc[t] <= Cmax, t=0..T
  0 <= ud[t] <= Dmax, t=0..T
end
```

FIGURE 9 CVXGEN code segment for energy storage. This figure shows the CVXGEN code segment for the storage example. The parameters can also be hard coded in the problem specification if their values are known ahead of time. For testing and simulation purposes, however, it is often better to declare the problem data as parameters so that they can be changed without having to regenerate the solver.

CONCLUSIONS

In this article we have shown that receding horizon control offers a straightforward method for designing feedback controllers that deliver good performance while respecting complex constraints. A designer specifies the RHC controller by specifying the objective, constraints, prediction method, and horizon, each of which has a natural choice suggested directly by the application. In more traditional approaches, such as PID control, a designer tunes the controller coefficients, often using trial and error, to handle the objectives and constraints indirectly. In contrast, RHC controllers can often obtain good performance with little tuning.

In addition to the straightforward design process, we have seen that RHC controllers can be implemented in real time at kilohertz sampling rates. These speeds are useful for both real-time implementation of the controller as well as rapid Monte Carlo simulation for design and testing

TABLE 2 Computer properties. This table summarizes the properties of the computers used to test the performance of the code generated by CVXGEN. The computers vary widely in speed, cache size, and power consumption.

	OS	Processor	Cache Size	Max Speed	Max Power
Computer 1	Linux 2.6	Intel Atom Z530	512 kB	1.60 GHz	2 W
Computer 2	Linux 2.6	Intel Core Duo T2300	2 MB	1.66 GHz	31 W
Computer 3	OS X 10.6	Intel Core i7-860	8 MB	3.46 GHz	95 W

purposes. Thus, receding horizon control can no longer be considered a slow, computationally intensive policy. Indeed, RHC can be applied to a wide range of control problems, including applications involving fast dynamics.

With advances in automatic code generation, RHC controllers can now be rapidly designed and implemented. The RHC optimization problem can be specified in a high-level description language, and custom solvers for the problem family can be automatically generated by a software tool, such as CVXGEN. The generated code is optimized for the specific problem family and is often orders of magnitude faster than a general optimization solver, such as Sedumi or SDPT3. In addition, the generated code has few external library dependencies, which facilitates implementation on different real-time platforms.

Receding horizon control combined with automatic code generation is a framework for designing and implementing feedback controllers. This framework allows designers with little optimization expertise to rapidly design and implement sophisticated high-performance controllers for a wide range of real-time applications.

AUTHOR INFORMATION

Jacob Mattingley is an electrical engineering Ph.D. student at the Information Systems Laboratory at Stanford University. He received the B.E. (Hons) degree in electrical and computer engineering from the University of Canterbury in 2005 and the M.S. degree in electrical engineering at Stanford University in 2007. He is currently working on automatic code generation, as well as computer modeling and engineering applications of convex optimization.

Yang Wang (yw224@stanford.edu) received B.A. and M.Eng. degrees in electrical and information engineering from Cambridge University in 2006. He is currently pursuing a Ph.D. degree in electrical engineering at the Information Systems Laboratory at Stanford University. His research interests include convex optimization with applications to control, signal processing, and machine learning. He is supported by a Rambus Corporation Stanford Graduate Fellowship. He can be contacted at Packard Electrical Engineering, Room 243, 350 Serra Mall, Stanford, California 94305 USA.

Stephen Boyd is the Samsung Professor of Engineering and professor of Electrical Engineering in the Information Systems Laboratory at Stanford University. He received the A.B. degree in mathematics from Harvard University in 1980 and the Ph.D. in electrical engineering and computer science from the University of California, Berkeley. He has been a faculty member at Stanford since 1985. His current research focus is on convex optimization applications in control, signal processing, and circuit design.

REFERENCES

[1] W. H. Kwon and S. Han, *Receding Horizon Control*. New York: Springer-Verlag, 2005.

[2] P. Whittle, *Optimization Over Time*. NY: Wiley, 1982.

[3] S. S. Keerthi and E. G. Gilbert, "Optimal infinite-horizon feedback laws for a general class of constrained discrete-time systems: Stability and moving-horizon approximations," *J. Optim. Theory Appl.*, vol. 57, no. 2, pp. 265–293, 1988.

[4] G. C. Goodwin, M. M. Seron, and J. A. De Doná, *Constrained Control and Estimation*. New York: Springer-Verlag, 2005.

[5] J. M. Maciejowski, *Predictive Control with Constraints*. Englewood Cliffs, NJ: Prentice-Hall, 2002.

[6] P. Scokaert and J. B. Rawlings, "Constrained linear quadratic regulation," *IEEE Trans. Automat. Contr.*, vol. 43, no. 8, pp. 1163–1169, Aug. 1998.

[7] J. B. Rawlings and D. Mayne, *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2009.

[8] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Eng. Pract.*, vol. 11, no. 7, pp. 733–764, 2003.

[9] E. G. Cho, K. A. Thoney, T. J. Hodgson, and R. E. King, "Supply chain planning: Rolling horizon scheduling of multi-factory supply chains," in *Proc. 35th Conf. Winter Simulation: Driving Innovation*, New Orleans, LA, 2003, pp. 1409–1416.

[10] F. Herzog. (2005). Strategic portfolio management for long-term investments: An optimal control approach. Ph.D. thesis. ETH, Zurich [Online]. Available: http://e-collection.ethbib.ethz.ch/view/eth:28198

[11] K. T. Talluri and G. J. Van Ryzin, *The Theory and Practice of Revenue Management*. New York: Springer-Verlag, 2004.

[12] M. J. West, C. M. Bingham, and N. Schofield, "Predictive control for energy management in all/more electric vehicles with multiple energy storage units," in *Proc. IEEE Int. Electric Machines and Drives Conf.*, Madison, WI, 2003, vol. 1, pp. 222–228.

[13] G. Stewart and F. Borrelli, "A model predictive control framework for industrial turbodiesel engine control," in *Proc. 47th IEEE Conf. Decision and Control*, Cancun, Mexico, 2008, pp. 5704–5711.

[14] S. Di Cairano, D. Yanakiev, A. Bemporad, I. Kolmanovsky, and D. Hrovat, "An MPC design flow for automotive control and applications to idle speed regulation," in *Proc. 47th IEEE Conf. Decision and Control*, Cancun, Mexico, 2008, pp. 5686–5691.

[15] P. Falcone, F. Borrelli, J. Asgari, E. H. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Trans. Contr. Syst. Technol.*, vol. 15, no. 3, pp. 566–580, 2007.

[16] R. Franz, M. Milam, and J. Hauser, "Applied receding horizon control of the Caltech ducted fan," in *Proc. American Control Conf.*, Anchorage, AK, 2002, pp. 3735–3740.

[17] S. Di Cairano and A. Bemporad, "Model predictive control tuning by controller matching," *IEEE Trans. Automat. Contr.*, vol. 55, no. 1, pp. 185–190, 2010.

[18] A. Bemporad and C. Filippi, "Suboptimal explicit receding horizon control via approximate multiparametric quadratic programming," *J. Optim. Theory Appl.*, vol. 117, no. 1, pp. 9–38, Nov. 2004.

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

[20] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," in *Proc. IFAC World Congress*, Seoul, South Korea, July 2008, pp. 6974–6997.

[21] M. Diehl, R. Findeisen, S. Schwarzkopf, I. Uslu, F. Allgöwer, H. G. Bock, E. D. Gilles, and J. P. Schlöder, "An efficient algorithm for nonlinear model predictive control of large-scale systems. Part I: Description of the method," *Automatisierungstechnik*, vol. 50, no. 12, pp. 557–567, 2002.

[22] 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.

[23] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior point methods to model predictive control," *J. Optim. Theory Appl.*, vol. 99, no. 3, pp. 723–757, Nov. 2004.

[24] J. E. Mattingley and S. Boyd. (2010, Apr.) CVXGEN: Automatic convex optimization code generation (web page and software) [Online]. Available: http://cvxgen.com/

[25] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[26] M. Grant, S. Boyd, and Y. Ye, "Disciplined convex programming," in *Global Optimization: From Theory to Implementation*, L. Liberti and N. Maculan, Eds. New York: Springer-Verlag, 2006, pp. 155–210.

[27] J. E. Mattingley and S. Boyd, "Real-time convex optimization in signal processing," *IEEE Signal Processing Mag.*, vol. 23, no. 3, pp. 50–61, 2009.

[28] J. E. Mattingley and S. Boyd, "Automatic code generation for real-time convex optimization," in *Convex Optimization in Signal Processing and Communications*, D. P. Palomar and Y. C. Eldar, Eds. Cambridge, U.K.: Cambridge Univ. Press, 2010, pp. 1–41.

[29] M. Diehl, H. G. Bock, and J. P. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM J. Control Optim.*, vol. 43, no. 5, pp. 1714–1736, 2005.

[30] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming," *Math. Program. A*, vol. 89, no. 1, pp. 149–185, 2000.

[31] A. Romanenko and L. O. Santos, "A nonlinear model predictive control framework as free software: Outlook and progress report," in *Assessment and Future Directions of Nonlinear Model Predictive Control*, R. Findeisen, F. Allgöwer, and L. Biegler, Eds. New York: Springer-Verlag, 2007, pp. 229–238.

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

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

[34] B. Houska and H. J. Ferreau. (2008, Aug.) ACADO toolkit: Automatic control and dynamic optimization (Web page and software) [Online]. Available: http://www.acadotoolkit.org/

[35] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in Identification and Control*, A. Garulli, A. Tesi, and A. Vicino, Eds. New York: Springer-Verlag, 1999, pp. 207–226.

[36] P. J. Goulart, E. C. Kerrigan, and J. M. Maciejowski, "Optimization over state feedback policies for robust control with constraints," *Automatica*, vol. 42, no. 4, pp. 523–533, 2006.

[37] M. Cannon, P. Couchman, and B. Kouvaritakis, "MPC for stochastic systems," in *Assessment and Future Directions of Nonlinear Model Predictive Control*, R. Findeisen, F. Allgöwer, and L. T. Biegler, Eds. New York: Springer-Verlag, 2007, pp. 255–268.

[38] M. Shin and J. A. Primbs, "A fast algorithm for stochastic model predictive control with probabilistic constraints," in *Proc. American Control Conf.*, Baltimore, MD, 2010, pp. 5489–5494.

[39] D. Q. Mayne, J. B. Rawlings, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[40] E. Camacho and C. Bordons, *Model Predictive Control*. New York: Springer-Verlag, 2004.

[41] Y. Wang and S. Boyd, "Performance bounds for linear stochastic control," *System Control Lett.*, vol. 53, no. 3, pp. 178–182, Mar. 2009.

[42] Y. Wang and S. Boyd. (2010). Fast evaluation of quadratic control—Lyapunov policy. *IEEE Trans. Contr. Syst. Technol.* [Online]. Available: http://www.stanford.edu/ boyd/papers/fast_clf.html

[43] J. A. Primbs, "A soft constraint approach to stochastic receding horizon control," in *Proc. 46th IEEE Conf. Decision and Control*, New Orleans, LA, 2007, pp. 4797–4802.

[44] J. Löfberg. (2004). YALMIP: A toolbox for modeling and optimization in MATLAB. presented at CACSD Conference, Taipei, Taiwan [Online]. Available: http://control.ee.ethz.ch/ joloef/yalmip.php

[45] M. Grant and S. Boyd. (2008, July). CVX: Matlab software for disciplined convex programming (Web page and software) [Online]. Available: http://www.stanford.edu/boyd/cvx/

[46] J. E. Mattingley and S. Boyd. (2008, Aug.). CVXMOD: Convex optimization software in Python (Web page and software) [Online]. Available: http://cvxmod.net/

[47] K. C. Toh, M. J. Todd, and R. H. Tütüncü, "SDPT3—A Matlab software package for semidefinite programming, version 1.3," *Optim. Methods Software*, vol. 11, no. 1, pp. 545–581, 1999.

[48] R. Tütüncü, K. Toh, and M. J. Todd, "Solving semidefinite-quadratic-linear programs using SDPT3," *Math. Program.*, vol. 95, no. 2, pp. 189–217, 2003.

[49] J. Sturm. (1999). Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optim. Methods Software* [Online]. 11, pp. 625–653. Available: http://sedumi.ie.lehigh.edu/

[50] A. Bemporad, "Model predictive control design: New trends and tools," in *Proc. 45th IEEE Conf. Decision and Control*, San Diego, CA, 2006, pp. 6678–6683.

[51] T. C. Miller, *Hierarchical Operations and Supply Chain Planning*. New York: Springer-Verlag, 2002.

[52] J. D. Schwartz, W. Wang, and D. E. Rivera, "Simulation-based optimization of process control policies for inventory management in supply chains," *Automatica*, vol. 42, no. 8, pp. 1311–1320, 2006.

[53] S. Bose and J. F. Pekny, "A model predictive framework for planning and scheduling problems: A case study of consumer goods supply chain," *Comput. Chem. Eng.*, vol. 24, no. 2–7, pp. 329–335, 2000.

[54] E. Mestan, M. Turkay, and Y. Arkun, "Optimization of operations in supply chain systems using hybrid systems approach and model predictive control," *Ind. Eng. Chem. Res.*, vol. 45, no. 19, pp. 6493–6503, 2006.

[55] E. Perea-Lopez, B. E. Ydstie, and I. E. Grossmann, "A model predictive control strategy for supply chain optimization," *Comput. Chem. Eng.*, vol. 27, no. 8–9, pp. 1201–1218, 2003.

[56] A. Gupta and C. D. Maranas, "Managing demand uncertainty in supply chain planning," *Comput. Chem. Eng.*, vol. 27, no. 8–9, pp. 1219–1227, 2003.

[57] S. Irani and K. Pruhs, "Algorithmic problems in power management," *SIGACT News*, vol. 36, no. 2, pp. 63–76, 2005.

[58] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware computer systems: Opportunities and challenges," *IEEE Micro*, vol. 23, no. 6, pp. 52–61, 2003.

[59] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *Proc. 33rd Int. Symp. Computer Architecture (ISCA'06)*, 2006, pp. 78–88.

[60] A. Wierman, L. L. H. Andrew, and A. Tang, "Power-aware speed scaling in processor sharing systems," in *Proc. IEEE INFOCOM*, Rio de Janeiro, Brazil, 2009, pp. 2007–2015.

[61] A. Mutapcic, S. Boyd, S. Murali, D. Atienza, G. De Micheli, and R. Gupta, "Processor speed control with thermal constraints," *IEEE Trans. Circuits Syst.*, vol. 56, no. 9, pp. 1994–2007, 2009.

[62] M. Houwing, R. R. Negenborn, B. De Schutter, P. W. Heijnen, and H. Hellendoorn, "Least-cost model predictive control of residential energy resources when applying μCHP," in *Proc. Power Tech*, July 2007, vol. 291, pp. 425–430.

[63] E. Handschin, F. Neise, H. Neumann, and R. Schultz, "Optimal operation of dispersed generation under uncertainty using mathematical programming," *Int. J. Elect. Power Energy Syst.*, vol. 28, no. 9, pp. 618–626, 2006.

[64] S. D. Braithwait, "Real-time pricing and demand response can work within limits," *Natural Gas Electricity*, vol. 21, no. 11, pp. 1–9, 2005.

[65] M. Khalid and A. V. Savkin, "Model predictive control for wind power generation smoothing with controlled battery storage," in *Proc. Joint IEEE Conf. Decision and Control and Chinese Control*, Shanghai, China, 2009, pp. 7849–7853.

[66] R. Kumar and B. Yao, "Model based power-split and control for electric energy system in a hybrid electric vehicle," in *Proc. IMECE 2006*, Chicago, IL, 2006, pp. 335–341.

[67] Z. Preitl, P. Bauer, B. Kulcsar, G. Rizzo, and J. Bokor, "Control solutions for hybrid solar vehicle fuel consumption minimization," in *Proc. 2007 IEEE Intelligent Vehicles Symp.*, Istanbul, Turkey, 2007, pp. 767–772.

[68] J. E. Mattingley and S. Boyd. (2010, Nov.). CVXGEN: A code generator for embedded convex optimization [Online]. Available: http://stanford.edu/ boyd/papers/code_gen_impl.html