2010 IEEE International Symposium on Computer-Aided Control System Design
Part of 2010 IEEE Multi-Conference on Systems and Control
Yokohama, Japan, September 8-10, 2010

Code Generation for Receding Horizon Control

Jacob Mattingley Yang Wang Stephen Boyd

Abstract— **Receding horizon control (RHC), also known as model predictive control (MPC), is a general purpose control scheme that involves repeatedly solving a constrained optimization problem, using predictions of future costs, disturbances, and constraints over a moving time horizon to choose the control action. RHC handles constraints, such as limits on control variables, in a direct and natural way, and generates sophisticated feed-forward actions. The main disadvantage of RHC is that an optimization problem has to be solved at each step, which leads many control engineers to think that it can only be used for systems with slow sampling (say, less than one Hz). Several techniques have recently been developed to get around this problem. In one approach, called explicit MPC, the optimization problem is solved analytically and explicitly, so evaluating the control policy requires only a lookup table search. Another approach, which is our focus here, is to exploit the structure in the optimization problem to solve it efficiently. This approach has previously been applied in several specific cases, using custom, hand written code. However, this requires significant development time, and specialist knowledge of optimization and numerical algorithms. Recent developments in convex optimization code generation have made the task much easier and quicker. With code generation, the RHC policy is specified in a high-level language, then automatically transformed into source code for a custom solver. The custom solver is typically orders of magnitude faster than a generic solver, solving in milliseconds or microseconds on standard processors, making it possible to use RHC policies at kilohertz rates. In this paper we demonstrate code generation with two simple control examples. They show a range of problems that may be handled by RHC. In every case, we show a speedup of several hundred times from generic parser-solvers.**

I. Introduction

Receding horizon control (RHC) or *model predictive control* (MPC) [1], [2], [3], [4] is a form of feedback control system that first became popular in the 1980s. With RHC, we solve an optimization problem at each time step to determine a plan of action over a fixed time horizon, and then apply the first input from this plan. At the next time step we repeat this, solving a new optimization problem, with the time horizon shifted one step forward. Estimates of future quantities, based on available measurements and data, enter the optimization; this provides *feedback* (*i.e.*, the use of real-time measurements or other information in determining the input).

RHC is a nonlinear control policy that naturally handles input constraints, output constraints, and a variety of control objectives. Systems can thus be controlled near their physical limits, obtaining performance superior to linear control. RHC has given excellent results in a wide range of practical settings, including industrial and chemical process control [5], supply chain management [6], stochastic control in economics and finance [7], and revenue management [8].

A drawback of RHC is the comparatively long time required to solve the planning problem using conventional numerical optimization techniques, as compared to, say, the time required to compute the control action in a traditional linear controller. Thus, RHC has been mostly limited to slow systems with sample times measured in seconds, minutes, or hours. Many methods have been proposed to speed up the solution of the optimization problems that arise in RHC. When the problem dimensions (the numbers of states, inputs, and constraints) are small, one approach is explicit MPC [9], [10], where symbolic solutions are generated offline and saved for later use. The online algorithm then reduces to a lookup table search, followed by a simple linear control law evaluation, which can be made extremely fast. Another method, applicable to a problem of any size, is to code custom online optimization solvers that exploit the particular problem structure that arises in RHC applications [11], [12], [13], [14]. These custom solvers can yield computation times that are several orders of magnitude faster than generic solvers, but require time-consuming hand coding, and significant expertise in optimization algorithms and numerical computation.

In this paper, we describe recent advances that make it much easier to develop custom RHC solvers. By combining a high-level specification language for optimization and recently-developed code generation tools, a user of RHC can quickly specify and generate fast, reliable custom code. Since the user does not require much optimization expertise, many more people can use RHC, and in new settings—including applications with kilohertz sample rates.

We do not claim that RHC, or any other modern control method, will always outperform traditional control methods. In many cases, a skilled designer can achieve similar performance by carefully tuning a conventional PID (proportional-integral-derivative) or other linear controller, suitably modified to handle the constraints. In our opinion, the main advantage of RHC is the simple design process, that handles constraints directly (indeed, by simply specifying them), and requires far less tweaking and adjustment than is typically required with conventional controller design. Roughly speaking, with RHC the designer simply lists the constraints, whereas in a conventional design process, the designer tweaks controller gains (or design weights in a modern method) to indirectly handle the constraints. We believe RHC via automatic code generation offers an attractive framework for rapid design of sophisticated controllers; especially for problems with challenging constraints, and even for problems with relatively fast sample rates.

In the remainder of the paper, we give a high-level

overview of RHC, briefly explain code generation for RHC using the software package CVXGEN [15], and illustrate the ideas with two examples. The examples are simple, and chosen to show the variety of problems that can be addressed. Our discussion will avoid unnecessary detail, so we refer the interested reader to [16] for a more detailed account of convex optimization, [17] for more on disciplined convex programming, and [18], [19] for a discussion of code generation for convex optimization.

We restrict our attention to systems with linear dynamics and convex objectives and constraints, for several reasons. First, many real systems can be reasonably modeled in this restricted form. Secondly, standard linearization techniques can be used to extend these methods to many nonlinear systems. (Indeed, almost all commercial MPC systems for process control rely on linearization around an operating point.) And finally, many of the techniques we discuss could be applied to general nonlinear systems. For some work in this area, see [20], which describes the software package NEWCON; an example of automatic code generation applied to nonlinear RHC [21], or a more recent approach by the same author applied to a two-link robot arm in [22]. Also see the ACADO system [23].

II. Formulating RHC problems

A. System dynamics and control

System dynamics. Each of the examples in this paper applies RHC to a discrete-time linear dynamical system of the form

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

where $x_t \in \mathbf{R}^n$ is the system state, $u_t \in \mathbf{R}^m$ is the control action or 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 they may change with time, but in many applications some of these data are constant and known.

Constraints and objective. The state and input must satisfy some constraints, 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. The instantaneous cost depends on both the current state and control action, and is denoted $\ell_t(x_t, u_t)$. We judge the quality of control by the average cost,

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

where we assume the limit exists. If $\ell_t(x_t, u_t)$ is a random variable, we replace $\ell_t(x_t, u_t)$ with $\mathbf{E}\,\ell_t(x_t, u_t)$. Like the dynamics data, we subscript the constraint set and objective function with the time t, to handle the case when they vary with time. But in many applications they are constant and known.

Information available for control. The control input u_t is determined using the information available to the controller at time t, including *estimates* of any quantities (or functions) that are not known, based on information that is known. We will denote these estimates as

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

where the notation $\hat{z}_{\tau|t}$ means our estimate of z_τ, based on information available to us at time t, where $\tau \geq t$. 'Information available at time t' includes conventional data in a control system, such as those available from sensor measurements, or known coefficients. It can also include other relevant information, such as historical usage patterns, weather, and price trends, which are not traditional data in control systems.

These estimates can be obtained in many ways. In the simplest case, we *know* the quantity being estimated, in which case we simply replace the estimates with the known value. For example, if the system dynamics matrices A_t and B_t have known and constant values A and B, we take $\hat{A}_{\tau|t} = A$ and $\hat{B}_{\tau|t} = B$. If the controller has access to the (exact) current state x_t, we take $\hat{x}_{t|t} = x_t$.

A traditional method for obtaining the estimates is from a statistical model of the unknown data, in which case the estimates can be conditional expectations or other statistical estimates, based on the data available at time t. For example, the additive terms c_t are often modeled as independent zero mean random variables, with natural estimate $\hat{c}_{\tau|t} = 0$.

The estimates need not be derived from statistical models; for example, future prices (entering the objective through ℓ_t, say) could be obtained from a futures market, or from analysts who predict trends. Another source of the estimates comes up when the system to be controlled is nonlinear. In this case $\hat{A}_{\tau|t}$, $B_{\tau|t}$, and $c_{\tau|t}$ can be a linearization of (nonlinear) dynamics, along a trajectory.

Controller design problem. The controller design problem is to find a control policy or control law that chooses the input u_t as a function of the quantities known at time t, in such a way that the constraints are always (or almost always) satisfied, and that the average cost J is minimized, or at least made small.

We have not fully specified the uncertainty model, so our description of the control problem is informal, and we cannot really talk about an optimal control policy. But when we give a full mathematical description of the uncertainty, for example as a statistical model, we can talk about the optimal control policy, *i.e.*, the policy that minimizes J, among all policies that map the information available into a control action while respecting the constraints.

B. Receding horizon control

The basic RHC policy is very simple. At time t, we consider an interval extending T steps into the future: $t, t+1, \ldots, t+T$. We then carry out several steps (which we briefly summarize, then describe again in more detail):

1) *Form a predictive model.* Replace all unknown quantities over the time interval with their current estimates, using all data available at time t.

2) *Optimize.* Solve the problem of minimizing the (predicted) objective, subject to the (predicted) dynamics and constraints.
3) *Execute.* Choose u_t to be the value obtained in the optimization problem of step 2.

Steps 1 and 2. The RHC optimization problem in step 2 takes the form

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

with variables $\hat{x}_t, \ldots, \hat{x}_{t+T+1}$ and $\hat{u}_t, \ldots, \hat{u}_{t+T}$. The data 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, $\hat{x}_{t|t}$. (In most applications, we can use known, exact values for many of the parameters.)

We can interpret $\hat{u}_t^\star, \ldots, \hat{u}_{t+T}^\star$, the optimal values from the RHC optimization problem (1), as a *plan of action* for the next T steps.

Step 3. We then choose $u_t = \hat{u}_t^\star$ to be our RHC action.

At the next time step, the process is repeated, with (possibly) updated estimates of the current state and future quantities. We make a few comments about the RHC policy.

Terminal constraints or cost terms. It is common to add a final state constraint, or an extra 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, with V a cost function for the final state. This can allow simpler, shorter-horizon controllers to approximate the behavior of controllers with longer horizons.

Optimality. The RHC policy is generally not an optimal control policy, even when we have a formal model of the uncertainty. Instead, RHC is merely a (sophisticated) heuristic which works very well in many applications.

Convexity. We will assume that $\mathcal{C}_t$ and ℓ_t are convex, which means that the RHC problem (1) is a convex optimization problem. This means that we can solve it efficiently, using standard convex optimization tools [16].

Requirements. To specify an RHC policy, we must describe the prediction method (*i.e.*, the method for estimating unkown quantities from current data), the horizon T, and any terminal cost or constraint.

C. Computer modeling of RHC

The two substantial design tasks required for RHC are system modeling, and creating the method to solve the optimization problem (1). The former task involves choosing a system model and appropriate cost functions and constraints, and has been tackled extensively in the literature [24], [25], [26]. A wide range of computing tools are available to accelerate development of that phase. That leaves the onerous task of solving (1) at an acceptable speed. This is needed during development and testing, and, especially, for implementation in a real-time system. Many convenient software tools [27], [28], [29] are available for solving convex problems, during development and testing. These *parser-solvers* take a high-level specification, and perform the necessary transformations for solution by a standard convex optimization solver, *e.g.*, [30], [31], [32]. This allows quick iteration, with the engineer able to change the optimization problem and immediately see results. However, during development, the demands on the software are not onerous, since an engineer is 'in the loop', and must formulate and evaluate each design. Thus, high speed or accuracy is not especially relevant.

Once a design is final and it is time for deployment, solver speed can be of critical importance, particularly when the sampling rate is high. If the solver speed is much faster than that required, we can use a less powerful processor, or a processor performing other tasks. While a parser-solver may be suitable for some, slow applications, they are usually unsuitable for online RHC. The traditional route is custom code development, either using a toolbox (see [11], or the references in [33]), or from scratch. This process is very difficult for most users, especially those without a numerical optimization background. Additionally, if the problem statement changes, even slightly, all code modifications must be made laboriously, by hand. Small changes in the formulation can lead to dramatic changes in the code. After that, testing and correcting the code must be done all over again.

Here we focus on an alternative: CVXGEN [15]. It combines a simple, natural modeling language with a code generator for general convex quadratic programs. It can be used much like a parser-solver, but instead of performing the transformation to and from the standard form just once, it generates code for performing both the transformation and the actual solution for a specific convex optimization problem family. This allows much faster design iteration, and can involve the code generator much earlier in the process. That means that even simulation and testing can take place at greatly accelerated speeds.

In the remainder of the paper, we describe two RHC application examples, giving CVXGEN code for each one. CVXGEN results are collected together in Table II in §IV.

III. Examples

A. Pre-ordering

Problem statement. We consider the problem of meeting a fluctuating demand for a perishable commodity, by pre-ordering it with different lead times and also purchasing it on the spot market, all at (possibly) different prices. When we place an order, we will specify delivery for between 1 and n periods in the future. (Here, 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 in period t, to be delivered in period $t+i$. Our state will be the order book $x_t \in \mathbf{R}_+^n$, where $(x_t)_i$ is the quantity scheduled to arrive in period $t+i-1$; in particular, $(x_t)_1$ is the stock at hand. The system dynamics are $x_{t+1} = Ax_t + Bu_t$, where A is a 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 our general model, we take $\mathcal{C}_t = \mathbf{R}^n \times \mathbf{R}^n_+$.) Thus, in this example, there is no uncertainty in the dynamics or constraints.

Our stage cost has two terms: The cost of placing orders for future delivery (which we recognize immediately), and the cost of making up any 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 price of ordering one unit of the commodity for delivery in period $t + i$. The unmet demand is $(d_t - (x_t)_1)_+$, where $d_t \geq 0$ is the demand in period t, and $(\cdot)_+$ denotes the positive part. The cost of meeting the excess demand on the spot market is $p_t^{\mathrm{spot}}(d_t - (x_t)_1)_+$, where $p_t^{\mathrm{spot}} \geq 0$ is the spot market price at time t. Thus the overall stage cost is

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

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

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

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

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

In period t, the controller has access to the current order book x_t, and the current and last N 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 autocovariances r_τ. The orders made in period t must be based on this information.

Receding horizon policy. Our RHC policy requires estimates of future stage cost, which depends on the (unknown) future demand. We will take

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

i.e., the exponential of the conditional mean of log demand, given the previous N demand values. (Since we know the current demand, we simply take $\hat{d}_{t|t} = d_t$.) Since we have assumed the demand is a stationary log-normal process, the conditional expectation of $\log d_\tau$ is an affine (linear plus constant) function of $\log d_t, \ldots, \log d_{t-N}$:

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

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 will also add a terminal constraint, $\mathbf{1}^T \hat{x}_{t+T+1} = n\, \mathbf{E}\, d_t$, where $\mathbf{E}\, d_t = \exp(\mu + r_0/2)$. This ensures that we won't run out the inventory at the end of the horizon to avoid paying for the commodity.

The RHC optimization problem (1) becomes

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

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

Constant order policy. We will compare the RHC policy with a simple policy: At each time t, we let $u_t = (0, \ldots, 0, \bar{u})$, *i.e.*, we order a constant amount with the maximum delivery time. We will use $\bar{u} = \mathbf{E}\, d_t = \exp(\mu + r_0/2)$, *i.e.*, we order with maximum lead-time (presumably, at the lowest price) an amount equal to the average demand.

1) Related work: Much work has been done on supply chain planning. For an overview of the field, though without the optimization component, see [34]. For the application of RHC to the supply chain, see [35], or [6] with covers multi-factory supply chains. In [36], the authors use extensive simulation of MPC to test sensitivity of various policies, while [37] explore various levels of decentralization. Finally, for supply chain optimization with mixed-integer constraints see [38], and for planning under uncertainty see [39].

2) Numerical example: Our example has $n = 5$ order lead times, with prices

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

with $\gamma = 0.7$. (Thus, we get a constant 30% discount for each period of lead time.) The demand process data are $\mu = 0$, and $r_\tau = 0.1(0.95^\tau)$. Our RHC controller will use horizon $T = 30$, and we estimate future demand using the last $N = 100$ demands.

Results. We simulate both the RHC and constant ordering policies for 1000 steps (with the same demand realization). The constant order policy incurs an average cost $J = 0.37$, while, as expected, the RHC policy performs considerably better, with an average cost $J = 0.28$. Some example trajectories are shown in Figure 2. We compare the costs incurred by the RHC policy (blue) and constant policy (red), over 500 time steps. The plots are (from top to bottom): demand (d_t), pre-order cost ($p^T u_t$), spot market cost ($p^{\mathrm{spot}}(d_t - (x_t)_1)_+$), and overall stage cost ($\ell(x_t, u_t)$).

In Figure 3 we show the actual (black) and predicted (blue) log-demand trajectories starting at $t = 220$. 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 can see that while the predicted trajectory captures the general trend, the prediction error is relatively large.

The CVXGEN code takes up to 250 μs to solve at each time step, which is 4000$\times$ faster than with plain CVX. This speed is far faster than would ever be required. However, this means that we could use extensive Monte-Carlo simulation to test different scenarios and ordering stratgies. Further computation performance details are collected in Table II.

```
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
```

Fig. 1: CVXGEN code segment for the pre-ordering example.

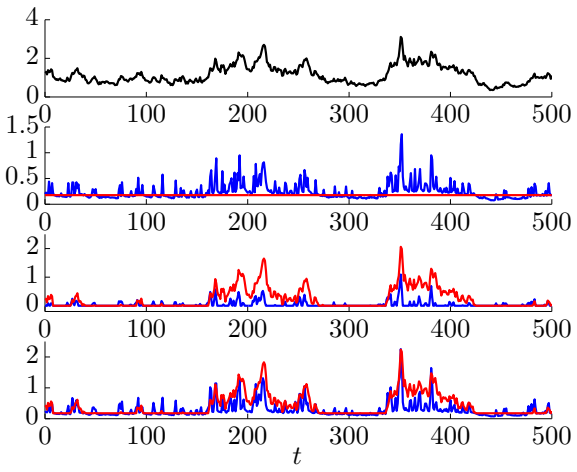

Fig. 2: Comparison of RHC policy (blue) and constant order policy (red) for the pre-order example. From top to bottom: demand (d_t), pre-order cost ($p^T u_t$), spot market cost ($p^{\text{spot}}(d_t - (x_t)_1)_+$), and stage cost ($\ell(x_t, u_t)$).

B. Energy storage

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

Let $q_t \geq 0$ denote the charge in the energy store at time period t. The energy store has capacity C, so we must have $q_t \leq C$. We let $u_t^c \geq 0$ denote the amount of energy taken from the source in period t to charge the energy store, and we let $u_t^d \geq 0$ denote the amount of energy discharged into the source from our energy store. (We will see that in each time period, at most one of these will be positive; that is, we will 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/discharge rates.

Charging increases the energy in our store by $\kappa^c u_t$, where $\kappa^c \in (0,1)$ is the charge efficiency; discharging decreases the energy in our store by u_t/κ^d, where $\kappa^d \in (0,1)$ is the discharge efficiency. In each time period 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 our general framework, 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 in period 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, we add a penalty of the form $\gamma(u_t^c + u_t^d)$, where $\gamma \geq 0$ is a parameter. (An alternative interpretation of this term is a transaction cost, with bid-ask spread γ: We buy energy at price $p_t + \gamma$, and sell energy back at price $p_t - \gamma$.) Our stage cost (*i.e.*, negative revenue, to be minimized) is thus

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

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

We will 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 period t the controller has access to the current charge level q_t, the data

$$C, \quad C^{\max}, \quad D^{\max}, \quad \kappa^c, \quad \kappa^d, \quad \eta, \quad \gamma,$$

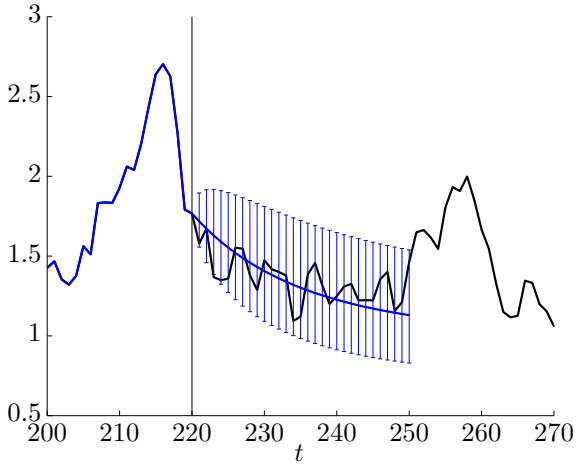

Fig. 3: Black: $\log d_t$; Blue: $\log \hat{d}_{t|220}$ for the pre-order example. Vertical lines show prediction error.

the current and last N 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.

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

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

which is an affine function of $\log p_t, \ldots, \log p_{t-N}$. (Note that this is not the same as $\mathbf{E}(p_\tau | p_t, \ldots, p_{t-N})$, which can also be computed and used as estimates of future prices.) Our estimates of the stage costs are

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

Thus, the RHC optimization problem becomes

$$\begin{array}{ll} \text{minimize} & \sum_{\tau=t}^{t+T} \hat{\ell}_t(\hat{q}_\tau, \hat{u}_t) \\ \text{subject to} & \hat{q}_{\tau+1} = \eta \hat{q}_\tau + \kappa^c \hat{u}^c_\tau - \hat{u}^d_\tau / \kappa^d, \\ & 0 \le \hat{u}^c_\tau \le C^{\max}, \quad 0 \le \hat{u}^d_\tau \le D^{\max}, \\ & \quad \tau = t, \ldots, t+T \\ & 0 \le \hat{q}_\tau \le C, \quad \tau = t, \ldots, t+T+1 \\ & \hat{q}_\tau = q_t, \end{array} \tag{2}$$

with variables $\hat{q}_t, \ldots, \hat{q}_{t+T+1}$, $\hat{u}^c_t, \ldots, \hat{u}^c_{t+T}$, $\hat{u}^d_t, \ldots, \hat{u}^d_{t+T}$. This is a convex optimization problem, and can be written as an LP.

Thresholding policy. We will compare the receding horizon policy with a simple thresholding policy, which works as follows:

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

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

In other words, we charge at the maximum rate if the price is below a threshold p_{thc}, and we discharge at the maximum rate if the price is above a threshold p_{thd}. If the price is in between we do nothing. We take the minimum to ensure we do not charge above the capacity or discharge below zero.

1) Related work: There is a particularly diverse set of work on optimization in energy storage and production. In [40], the authors consider a distributed energy system where individual grid-connected households use an MPC-based controller to control 'micro combined heat and power' plants. For more on distributed generation and variable pricing, see, respectively, [41] and [42]. On the generation side, [43] considers using MPC and batteries to smooth the power produced by wind turbines. The paper includes a case study with real data.

A different, but related application is for hybrid vehicles. Here multiple power sources are available. See [44] or [45], or for a vehicle with multiple different energy storage units see [46].

2) Numerical example: We look at a particular numerical instance with $\eta = 0.98$, $\kappa^c = 0.98$, $\kappa^d = 0.98$, $C^{\max} = 10$, $D^{\max} = 10$, $C = 50$, $\gamma = 0.02$, $q_0 = 0$, $\mu = 0$, $r_\tau = 0.1(0.99^\tau \cos(0.1\tau))$. For the receding horizon policy we used a time horizon of $T = 50$ steps, and $N = 100$ previous prices to estimate future prices.

```
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
```

Fig. 4: CVXGEN code segment for the storage example.

Results. The simulations were carried out for 1000 time steps. Figure 5 shows the cumulative profit,

$$r_t = \sum_{\tau=0}^{t} p_\tau(u^d_\tau - u^c_\tau) - \gamma(u^d_\tau + u^c_\tau),$$

for the RHC policy (blue) and the simple thresholding policy (red), over 500 time steps. For the thresholding policy, we adjusted the charge/discharge thresholds via trial and error to achieve good performance. The final thresholds we used are $p_{\text{thc}} = 0.8$, $p_{\text{thd}} = 1.3$. Clearly, the RHC policy outperforms the thresholding policy. The average profit achieved for the RHC policy is 0.23 per-period, whereas thresholding achieves a profit of 0.029 per-period (averaged over 1000 time steps).

Figure 6 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 takes up to 360 μs to solve at each time step, which is 3500$\times$ faster than with CVX. Again, these speeds are much faster than is required in practice, since prices would not usually vary on the time scale of microseconds. However, these computation speeds are useful for Monte Carlo simulations and scenario testing. Further computation performance details are collected in Table II.

IV. CVXGEN PERFORMANCE

To give a rough guide to CVXGEN's performance, we tested CVXGEN code for each example on three different computers. The given timings should not be taken too seriously, since there are many things that could easily improve performance, often reducing speed by an order of magnitude

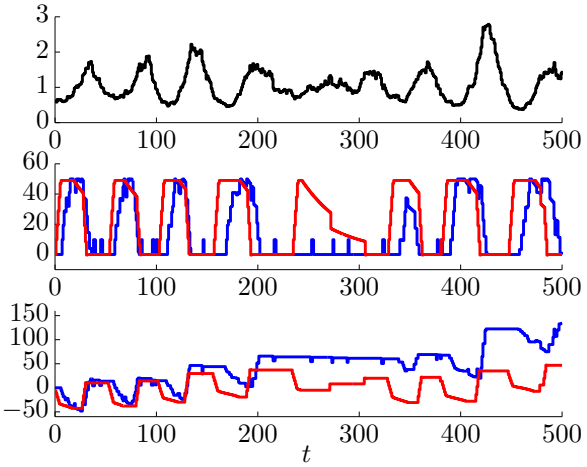

Fig. 5: Comparison of RHC policy (blue) and thresholding policy (red) for the storage example. From top to bottom: price (p_t), charge (q_t), cumulative profit (r_t).

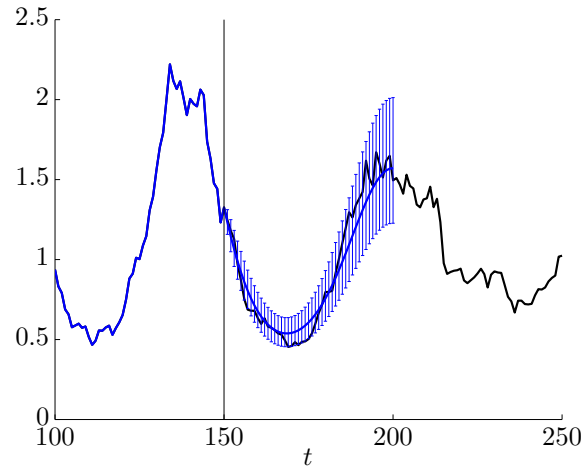

Fig. 6: Black: $\log p_t$; Blue: $\log \hat{p}_{t|150}$ for the storage example. Vertical lines show prediction error.

or more. First, single-precision floats could be used in place of double-precision, since the scale of data is known ahead of time. This would improve performance on a variety of processors. Secondly, the time-horizon selected for the given examples is relatively long. With a suitable choice of final state cost as in [47], this could be reduced further, giving a linearly proportional performance improvement. Finally, we solve the problems to high accuracy (so that control performance does not suffer from suboptimality), which required up to 16 steps. With a small amount of tuning, adequate control performance could easily be achieveable using a fixed step limit of (say) 5 steps [11]. Thus, all of the numerical results should be taken as preliminary upper bounds on performance, and they will change over time.

Each computer's properties are summarized in Table I. We used `gcc-4.4` on each processor, with the compiler optimization flag `-Os`. We have not yet conducted tests with

	OS	Processor, cache	Clock	Power
1	Linux 2.6	Intel Atom, 512 kB	1.60 GHz	2 W
2	Linux 2.6	Intel Core Duo, 2 MB	1.66 GHz	31 W
3	OS X 10.6	Intel Core i7, 8 MB	3.46 GHz	95 W

TABLE I: Computer properties

	`preorder`	`storage`
CVX and Sedumi (ms)	971	1290
Variables, original	310	153
Variables, transformed	341	153
Constraints, transformed	373	357
KKT matrix nonzeros	1116	1121
KKT factor fill-in	1.64	1.45
Max steps required	10	16
CVXGEN, Computer 1 (ms)	2.34	4.01
CVXGEN, Computer 2 (ms)	0.96	1.98
CVXGEN, Computer 3 (ms)	0.25	0.36

TABLE II: CVXGEN performance

a real-time operating system.

In each case, we ensured the computer was idle, then solved optimization problem instances continuously for at least one second. We calculated the maximum time taken for solving any instance, ensuring that each problem was solved to within 0.5% of optimality. For a rough guide to the speed of a traditional parser-solver (a somewhat unfair comparison), we also tested the performance of CVX and Sedumi on the fastest computer, Computer 3, using Matlab 7.9, CVX 1.2 and Sedumi 1.2. Results are summarized in Table II.

V. Conclusion

This paper has shown two examples of code generation in practice. In all cases we implemented a RHC policy, formulating it as a convex optimization problem. We then used CVXGEN to generate high-speed solvers specific to those problems, and demonstrated typical results. In situations like these, automatic code generation and RHC combine to make a control system designer's job easy and efficient. Significant performance improvements are possible as well.

References

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

[2] P. Whittle, *Optimization over Time*. John Wiley & Sons, Inc. New York, NY, USA, 1982.

[3] G. C. Goodwin, M. M. Seron, and J. A. De Doná, *Constrained control and estimation*. Springer, 2005.

[4] J. M. Maciejowski, *Predictive Control with Constraints*. Prentice Hall, 2002.

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

[6] 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 *Proceedings of the 35th conference on Winter simulation: driving innovation*, 2003, pp. 1409–1416.

[7] F. Herzog, "Strategic portfolio management for long-term investments: An optimal control approach," Ph.D. dissertation, ETH, Zurich, 2005.

[8] K. T. Talluri and G. J. V. Ryzin, *The Theory and Practice of Revenue Management*. Springer, 2004.

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

[10] 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.

[11] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," in *Proceedings IFAC World Congress*, Jul. 2008, pp. 6974–6997.

[12] 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.

[13] M. A. kerblad and A. Hansson, "Efficient solution of second order cone program for model predictive control," *International Journal of Control*, vol. 77, no. 1, pp. 55–77, Jan. 2004.

[14] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior point methods to model predictive control," *Journal of optimization theory and applications*, vol. 99, no. 3, pp. 723–757, Nov. 2004.

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

[16] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[17] M. Grant, S. Boyd, and Y. Ye, "Disciplined convex programming," in *Global Optimization: from Theory to Implementation*, ser. Nonconvex Optimization and Its Applications, L. Liberti and N. Maculan, Eds. New York: Springer Science & Business Media, Inc., 2006, pp. 155–210.

[18] J. E. Mattingley and S. Boyd, "Real-time convex optimization in signal processing," *To appear, IEEE Signal Processing Magazine*, 2009.

[19] ——, "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 University Press, 2010, pp. 50–61.

[20] A. Romanenko and L. O. Santos, "A nonlinear model predictive control framework as free software: outlook and progress report," *Assessment and Future Directions of Nonlinear Model Predictive Control*, pp. 229–238, 2007.

[21] T. Ohtsuka and A. Kodama, "Automatic code generation system for nonlinear receding horizon control," *Transactions of the Society of Instrument and Control Engineers*, vol. 38, no. 7, pp. 617–623, Jul. 2002.

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

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

[24] R. Johansson, *System modeling and identification*. Prentice Hall, 1993.

[25] L. Ljung and E. J. Ljung, *System identification: theory for the user*. Prentice Hall, 1987.

[26] O. Nelles, "Nonlinear system identification," *Measurement Science and Technology*, vol. 13, 2002.

[27] J. Löfberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004, http://control.ee.ethz.ch/~joloef/yalmip.php.

[28] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming (web page and software)," http://www.stanford.edu/~boyd/cvx/, Jul. 2008.

[29] J. E. Mattingley and S. Boyd, "CVXMOD: Convex optimization software in Python (web page and software)," http://cvxmod.net/, Aug. 2008.

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

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

[32] J. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization Methods and Software*, vol. 11, pp. 625–653, 1999, software available at http://sedumi.ie.lehigh.edu/.

[33] A. Bemporad, "Model predictive control design: New trends and tools," in *Proceedings of 45th IEEE Conference on Decision and Control*, 2006, pp. 6678–6683.

[34] T. C. Miller, *Hierarchical operations and supply chain planning*. Springer Verlag, 2002.

[35] 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.

[36] S. Bose and J. F. Pekny, "A model predictive framework for planning and scheduling problems: a case study of consumer goods supply chain," *Computers and Chemical Engineering*, vol. 24, no. 2-7, pp. 329–335, 2000.

[37] 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.

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

[39] A. Gupta and C. D. Maranas, "Managing demand uncertainty in supply chain planning," *Computers & Chemical Engineering*, vol. 27, no. 8-9, pp. 1219–1227, 2003.

[40] M. Houwing, R. R. Negenborn, B. D. S. P. W. Heijnen, and H. Hellendoorn, "Least-cost model predictive control of residential energy resources when applying μCHP," *Proceedings of Power Tech*, vol. 291, Jul. 2007.

[41] E. Handschin, F. Neise, H. Neumann, and R. Schultz, "Optimal operation of dispersed generation under uncertainty using mathematical programming," *International Journal of Electrical Power & Energy Systems*, vol. 28, no. 9, pp. 618–626, 2006.

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

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

[44] R. Kumar and B. Yao, "Model based power-split and control for electric energy system in a hybrid electric vehicle," in *Proceedings of IMECE 2006*, 2006.

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

[46] 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 *IEEE International Electric Machines and Drives Conference*, vol. 1, 2003.

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