On an Anytime Algorithm for Control

Vijay Gupta

Abstract—We present an algorithm to calculate the control input when the processing resources available are time-varying. The basic idea is to calculate the components of the control input vector sequentially, in order to maximally utilize the available processing resources at every time step. Alternatively, the algorithm can be viewed as using a sequence of increasingly complicated but accurate models of the process to refine the control input. For the LQG case, we provide analytical performance and stability expressions. For more general cases, we present a receding horizon control based implementation and indicate through numerical simulations that the increase in performance due to the proposed algorithm can be significant.

I. INTRODUCTION

With processing and communication devices decreasing in size and cost, a lot of attention has recently been focused on networked and embedded control (see, e.g., the special issues [1], [2] and the references therein). As more and more objects are equipped with micro-processors that are responsible for multiple functions such as control, communication, data fusion, system maintenance and so on, the implicit assumption traditionally made in control systems about the presence of a dedicated processor that is able to execute any desired control algorithm will break down. Similarly, if a remote controller controls many devices, multiple control tasks will compete for shared processor resources, leading to constrained availability of processing resources for any particular control loop. It is, thus, of interest to study control in the presence of limited and time-varying availability of processing power.

Owing to its importance, there are a growing number of works in this area. The impact of finite computational power has been looked at most closely for techniques such as receding horizon control (RHC). McGovern and Feron [13], [14] presented bounds on computational time for achieving stability for specific optimization algorithms, if the processor has constant, but limited, computational resources. Henriksson et al [10], [11] studied the effect of not updating the control input in continuous time systems for the duration of the computational delay for optimization algorithms based on active set methods. Another stream of work is the development of anytime algorithms that provide a solution even with limited processing resources, and refine the solution as more resources become available. Such algorithms can tolerate fluctuating processor availability, and are, thus, very popular in real-time systems. In control, however, there are very few methods available for developing anytime controllers, except for the notable works of Bhattacharya et al [3] and Greco et al [8]. Related streams of work in this area also include event-triggered and self-triggered control systems [17], [18] that focus on infrequent computations (and, hence, better processor utilization for the same control performance), and scheduling of control tasks [4], [5], [16] that look at the problem of processor queue scheduling, when control calculation is merely one of the tasks in the queue.

In this paper, we concentrate on the following question: how do we guarantee good control performance, when the processor availability is time-varying in an a priori unknown fashion? To solve this problem, we will develop an anytime control algorithm, that will guarantee better performance as more processing time is available. For the Linear Quadratic Gaussian (LQG) case, we will be able to provide analytic stability and performance conditions. For more general cases, we will extend the algorithm to a receding horizon control formulation and illustrate the performance gain numerically.

The paper is organized as follows. We begin in Section II by formulating the problem, and stating the assumptions. In Section III, we present the proposed algorithm, and analyze the performance. The algorithm for the unconstrained LQG case and the associated stability and performance results are provided in Section III-A. We then illustrate how to extend the algorithm to the case when the state and control inputs are constrained, by using an RHC formulation in Section III-B. We numerically illustrate the improvement in performance using the proposed algorithm in Section III-C.

II. PROBLEM FORMULATION

Process Model: In this paper, we focus on linear processes. Consider the process

\[ x(k+1) = Ax(k) + Bu(k) + w(k), \quad x(0) \]  

with the state \( x(k) \in \mathbb{R}^n \), the control input \( u(k) \in \mathbb{R}^m \) and the process noise \( w(k) \) that is modeled as Gaussian and white with zero mean and covariance \( R_w \). The state and control input may additionally have to satisfy constraints of the form \( h(x(k), u(k)) \in S \) for some set \( S \). If such constraints are absent, we will refer to the problem as unconstrained; otherwise the problem will be referred to as being constrained. The control input needs to be calculated to minimize a non-negative cost function of the form

\[ J(x(0), x(1), \ldots, x(N_h+1), u(0), u(1), \ldots, u(N_h)). \]

\( N_h \) refers to the problem horizon. In this paper, we focus on the quadratic cost function, which is one of the most
Because of the randomness introduced by the execution time availability, the value of the cost function that is achieved becomes stochastic. It is, thus, useful to consider some moment of the cost function to characterize algorithms. We will consider the expected value of the achieved cost $E[J_{N_h}]$, where the expectation for the cost in function (2) is further taken over the sequence $\{\tau(k)\}$.

**Problem Description:** Given the probability mass function for the available execution time $\tau(k)$, we wish to design a control algorithm to optimize the expected performance $E[J_{N_h}]$. In the case when constraints on $x(k)$ and $u(k)$ are present, algorithms to optimize $E[J_{N_h}]$ are not available in general, even without additional processing resource limitations. Thus, in such cases, we may be interested merely in stabilizing the plant. As part of the problem formulation, it is also important to specify the value of the control input utilized if the controller is unable to calculate a new control input $u(k)$ at time $k$. Several choices can be made, including applying zero control, the control input $u(k-1)$, and so on. As Schenato [15] has pointed out, different choices may be optimal for different plant parameter values. For sake of concreteness, we assume that if the new control input cannot be calculated, zero control input ($u(k) = 0$) is applied.

### III. Proposed Algorithm

#### A. Unconstrained Systems

We begin by considering systems that do not have any constraints of the form $h(x(k)), u(k) \in \mathcal{S}$ as defined in the problem formulation.

1) **Description:** The algorithm is based on the following basic idea. The execution time $f(p)$ required to calculate the control input is an increasing function of the number of control inputs that are calculated. For a linear system, the control input is typically calculated as a linear function of the state. If the entire control vector is calculated, $m$ variables need to be calculated, thus requiring an execution time equal to $f(m)$. Instead, the algorithm that we propose calculates the components of the control input vector one at a time. While calculating the $t$-th component, the values of the first $t-1$ components are known from previous optimizations. Thus, as more execution time becomes available, the control input becomes progressively refined. In keeping with the assumption stated earlier, the value of the components of the control input that cannot be calculated is assumed to be zero. An alternate view of the algorithm as considering a sequence of increasingly accurate models is provided later.

For pedagogical ease, we present the algorithm below for the case when $m = 2$. Thus $u(k)$ is a two-dimensional vector with components $u_1(k)$ and $u_2(k)$. To identify the component that is calculated first, we need to prioritize among the components. We choose the following protocol: For each input $u_i$, write the process equation (1) in the controllable canonical form. For those modes that are not controllable using the input $u_i$, denote the maximum rate of increase of those modes by $\rho_i$. $\rho_i$ can be calculated as the spectral radius of the uncontrollable part of the system matrix $A$ from input $u_i$. If $\rho_1 < \rho_2$, $u_1$ is calculated before
The intuition behind this protocol is that if the modes that can only be controlled using $u_1$ are more unstable than those that can only be controlled using $u_2$, then $u_1$ needs to be calculated at a greater frequency than $u_2$.

Assume without generality that $p_1 < p_2$. We now proceed to rewrite (1) in the controllable canonical form with respect to $u_1$ as

$$
\begin{bmatrix}
    x_1(k+1) \\
    x_2(k+1)
\end{bmatrix} =
\begin{bmatrix}
    A_{11} & 0 \\
    A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
    x_1(k) \\
    x_2(k)
\end{bmatrix}
+ \begin{bmatrix}
    B_{11} & B_{12} \\
    0 & B_{22}
\end{bmatrix}
\begin{bmatrix}
    u_1(k) \\
    u_2(k)
\end{bmatrix} + Mw(k),
$$

where $x_1(.)$ and $x_2(.)$ are the modes that are, respectively, controllable and uncontrollable using $u_1(k)$. There are three cases that are possible:

1) If $\tau(k) < f(1)$, there isn’t enough execution time to calculate either of the control inputs. Thus, the control input $u_1(k) = u_2(k) = 0$ is calculated.

2) If $2f(1) > \tau(k) > f(1)$, only $u_1(k)$ can be calculated. The inputs $u_2(k)$ are assumed to be equal to 0.

3) If $\tau(k) > 2f(1)$, then the control value $u_2(k)$ can also be calculated along with the value of $u_1(k)$ calculated in the previous step.

To calculate the control vector and to show that $f(p)$ is indeed linear, we proceed as follows. Suppose $2f(1) > \tau(k) > f(1)$ with a probability $p_1$, and $\tau(k) > 2f(1)$ with a probability $p_2$. Then, using the above algorithm, the system evolves as

$$
x(k+1) =
\begin{cases}
    \begin{bmatrix}
        A_{11} & 0 \\
        A_{21} & A_{22}
    \end{bmatrix}
    x(k) + Mw(k)
    & \text{with prob. } 1 - p_1 - p_2 \\
    \begin{bmatrix}
        A_{11} & 0 \\
        A_{21} & A_{22}
    \end{bmatrix}
    x(k) + \begin{bmatrix}
        B_{11} \\
        0
    \end{bmatrix}
    u_1(k)
    & \text{with prob. } p_1 \\
    \begin{bmatrix}
        A_{11} & 0 \\
        A_{21} & A_{22}
    \end{bmatrix}
    x(k) + \begin{bmatrix}
        B_{11} \\
        0
    \end{bmatrix}
    u_1(k)
    & \text{with prob. } p_2
\end{cases}
$$

To obtain the control laws, we utilize the framework of Markovian jump linear systems (MJLS). The system described by the evolution in equation (3) is an MJLS with three modes, one corresponding to each of the three cases. Denote the three cases by mode 1, 2 and 3, respectively. For the quadratic cost as given in equation (2), it is standard (see, e.g., [6]) that the optimal control for MJLS is given by a linear function of the state. Thus, for some matrices $K_1(k)$ and $K_2(k),$

$$
\begin{align*}
    u(k) &= \begin{bmatrix}
        K_1(k)x(k) \\
        0
    \end{bmatrix} \quad \text{in mode 2} \\
    u(k) &= \begin{bmatrix}
        0 \\
        K_2(k)x(k)
    \end{bmatrix} \quad \text{in mode 3},
\end{align*}
$$

Further, if we denote

$$
\hat{A} = \begin{bmatrix}
    A_{11} & 0 \\
    A_{21} & A_{22}
\end{bmatrix}, \quad B_1 = \begin{bmatrix}
    B_{11} \\
    0
\end{bmatrix}, \quad B_2 = \begin{bmatrix}
    B_{12} \\
    B_{22}
\end{bmatrix}
$$

and $\bar{A} = \hat{A} + B_1 \begin{bmatrix}
    K_1(k) \\
    0
\end{bmatrix}$, then, it can be calculated using standard dynamic programming arguments that the matrices $K_1(k)$ and $K_2(k)$ that minimize the cost (2) are given by

$$
\begin{align*}
    K_1(k) &= -(R(1,1) + B_1^T V(k)B_1)^{-1} B_1^T V(k) \bar{A} \\
    K_2(k) &= -(R(2,2) + B_2^T V(k)B_2)^{-1} B_2^T V(k) \bar{A} \\
    V(k) &= p_1 \left( \bar{A}^T V(k+1) \bar{A} + Q - \bar{A}^T V(k+1) \bar{A} \right) \\
    &\quad + p_2 \left( \bar{A}^T V(k+1) \bar{A} + Q - \bar{A}^T V(k+1) \bar{A} \right) \\
    &\quad + (1 - p_1 - p_2) \left( \bar{A}^T V(k+1) \bar{A} + Q \right),
\end{align*}
$$

and $V(N_h + 1) = P_{N_h + 1}$.

Remarks: 1. The above algorithm can clearly be extended to the case when $m > 2$.

2. The above algorithm has the anytime property in the sense that the control input $u(k)$ is successively refined as more execution time becomes available. However, the quality of the input is not a continuous function of the available time.

3. Since the processor does not know the value of $\tau(k)$ a priori, the value of $u_1(k)$ is assumed to be already calculated while the value of $u_2(k)$ is calculated. If we assume that the processor knows $\tau(k)$ a priori, the third mode in (3) would be replaced by calculating $u_1(k)$ and $u_2(k)$ jointly for the case when $\tau(k) > f(2)$.

4. The values $V(k)$ can be computed and stored a priori. Only the terms $K_1(k)x(k)$ and $K_2(k)x(k)$ need to be computed online. If we adopt the same model of computation complexity as [3], we see that the time required $f(p)$ would be linear in the number of components $p$ being calculated. For other models, $f(p)$ may be a different function of $p$. However, the algorithm developed above can still be used.

2) Analysis: To benchmark the improvement in performance using the proposed algorithm, we consider an alternative algorithm that does not have the anytime property. In the baseline algorithm $A_1$, the processor calculates the optimal control law for a process evolving as in equation (1) whenever the execution time is sufficient (i.e., when $\tau(k) > f(2)$) and applies zero input otherwise. The process then evolves as another MJLS given by

$$
x(k+1) =
\begin{cases}
    Ax(k) + w(k) & \text{with prob. } 1 - q \\
    Ax(k) + Bu(k) + w(k) & \text{with prob. } q,
\end{cases}
$$

where, e.g., $q = p_2$ if $f(p)$ is linear in $p$. The optimal control input can again be calculated using MJLS theory.

Proposition 3.1: Consider the problem formulation as stated above with the baseline algorithm $A_1$ being used. Then, a necessary condition for stability is that the inequality
\[(1-q)\rho(A)^2 < 1\] is true where \(\rho(A)\) is the spectral radius of \(A\). Moreover, a condition that is both necessary and sufficient is that there exists a matrix \(K\) and a positive definite matrix \(P\) such that
\[
P > Q + (1-q) \left( A^T P A \right) + q \left( (A + BK)^T P (A + BK) + K^T R K \right) \tag{6}
\]
Finally, the cost \(E[J_{N_h}]\) is given by \(x^T(0)T(0)x(0)\), where \(T(0)\) is calculated using the recursion
\[
T(k) = A^T T(k+1) A + Q - q A^T T(k+1) B (R + B^T T(k+1) B)^{-1} B^T T(k+1) A \tag{7}
\]
with \(T(N_h+1) = P_{N_h+1}\).

Intuitively, this condition makes sense since a control input is calculated with probability \(q\) at every time step for the state \(x(k)\) that evolves at a rate \(\rho(A)\). Using similar tools, the performance of the system with the proposed algorithm can also be characterized.

**Proposition 3.2:** Consider the problem formulation as stated above with the proposed anytime algorithm being used. Then, a necessary condition for stability is that the following inequalities are true
\[
(1-p_1-p_2)\rho(A)^2 < 1, \quad (1-p_2)\rho(A_{22})^2 < 1,
\]
where \(\rho(A_{22})\) is the spectral radius of the matrix block \(A_{22}\).

Moreover, a condition that is both necessary and sufficient is that there exists matrices \(K_1\) and \(K_2\), and a positive definite matrix \(P\) such that
\[
P = p_1 \left( (\bar{A} + B_1 K_1)^T P (\bar{A} + B_1 K_1) + K_1^T R (1, 1) K_1 \right) \\
+ p_2 \left( (\bar{A} + B_2 K_2)^T P (\bar{A} + B_2 K_2) + K_2^T R (2, 2) K_2 \right) \\
+ (1-p_1-p_2) \left( A^T P A \right) + Q. \tag{8}
\]

The expected cost \(E[J_{N_h}]\) is given by \(x^T(0)V(0)x(0)\), where \(V(0)\) is calculated using the recursion (4).

The necessary conditions for stability are again intuitive. The first inequality arises because with a probability \(1-p_1-p_2\), no control input is calculated for modes that are increasing at a rate \(\rho(\bar{A})\), which is equal to \(\rho(A)\). The second inequality arises because the stabilizing control input for the modes \(x_{22}(k)\) that evolve at a rate \(\rho(A_{22})\) is only calculated at any time step with a probability \(p_2\).

**B. Constrained Systems**

We can extend the above idea to obtain anytime control algorithms to systems that have additional constraints of the form \(h(x(k), u(k)) \in S\) for some set \(S\), as described in the problem formulation. This case is significantly more difficult even without considering limitations on computational resources since for general sets \(S\), there are no systematic methods to design the control laws for constrained systems, similar to the Riccati-based design for unconstrained systems. We focus on the receding horizon control (RHC) methodology for this purpose. Moreover, analysis of performance is not available even without processing limitations. We will use some numerical simulations to illustrate the benefit of the anytime control algorithm.

1) **Receding Horizon Control**: Receding horizon control is one of the most powerful methods to calculate the control input for constrained systems. In this formulation, an optimization problem \(O\) of the form
\[
\min_{\{u(k+i)\}_{i=0}^{N}} \left\{ C_N \left(x(k), x(k+1), \ldots, x(k+N), \right) \right\} \tag{9}
\]
\[
u(k), u(k+1), \ldots, u(k+N) \right) \tag{10}
\]
\[
h(x(k+i), u(k+i)) \in S \forall i = 0, \ldots, N \tag{11}
\]
\[
x(k+N) \in T, \tag{12}
\]
for a suitable cost function \(C_N > 0\) is solved at every time step \(k\). The control input \(u(k)\) is then applied to the process and the optimization problem is solved again at time \(k+1\). The process then continues till the problem horizon \(N_h\). While, in general, \(C_N\) may not have any resemblance to the cost function \(J\), typically the functional form of \(C_N\) is assumed to be the same as that of \(J\). Thus, a form extremely common among practitioners is \(C_N\) being the quadratic cost \(J_N\) of the form given in (2). The additional constraint of the form \(x(k+N) \in T\) is sometimes used to ensure stability. Alternatively, one may impose the terminal weight \(P_{N+1}\) in the quadratic cost \(J_N\) to be sufficiently large. In the optimization problem stated above, \(N\) is referred to as the receding control horizon. Specifying the optimization problem \(O\) is referred to as designing the receding horizon control algorithm.

2) **Algorithm Description**: The algorithm once again proceeds by calculating the components of the control input sequentially. We begin by prioritizing among the inputs \(u_1(k)\) and \(u_2(k)\) as in Section III-A.1. Given the component of the control input with the highest priority, we convert the process into the controllable canonical form corresponding to that input, as in equation (3). The algorithm then proceeds as follows.

1) If \(\tau(k) < f(N)\), there isn’t enough execution time to solve the problem \(O\) for even a scalar control input. Thus, the control input \(u_1(k) = u_2(k) = 0\) is calculated.

2) If \(2f(N) > \tau(k) > f(N)\), the optimization problem \(O\) is solved by assuming \(\{u_1(j)\}_{j=k}^{k+N}\) to be the only optimization variables. The inputs \(\{u_2(j)\}_{j=k}^{k+N}\) are assumed to be equal to 0. Thus, the evolution equation that is used in place of (10) in the problem \(O\) is
\[
x(k+1) = \hat{A}x(k) + \begin{bmatrix} B_{11} \\ 0 \end{bmatrix} u_1(k) + M w(k). \tag{13}
\]

3) If \(\tau(k) > 2f(N)\), then the optimization problem \(O\) is again solved by assuming \(\{u_2(j)\}_{j=k}^{k+N}\) to be the optimization variables. The problem is solved by assuming that the inputs \(\{u_1(j)\}_{j=k}^{k+N}\) have already been calculated in the previous step. Thus, the equation
that is used in place of (10) is

\[
x(k + 1) = \hat{A}x(k) + \begin{bmatrix} B_{11} \\ 0 \end{bmatrix} u_1(k) \\
+ \begin{bmatrix} B_{12} \\ B_{22} \end{bmatrix} u_2(k) + Mw(k),
\]

where the values \( \{u_1(j)\}_{j=k}^{k+N} \) are known.

a) Remarks:

1) The above algorithm can be generalized to consider an arbitrary dimension \( m \) of the control vector. The elements of the vector are calculated one at a time, as above.

2) Assume that the cost function \( J_t \) is such that it does not couple the states \( x_1 \) and \( x_2 \) (in other words, if the matrices \( Q \) and \( R \) are block diagonal). Then, the reduced dimensional equation

\[
x_1(k + 1) = A_{11}x_1(k) + B_{11}u_1(k) + M_1w_1(k)
\]

can be used in place of equation (13) in the problem \( O \), for the case \( 2f(N) > \tau(k) > f(N) \). This yields an alternate interpretation of the proposed algorithm in terms of the fidelity of the model of the process (1).

As more execution time is available at the processor, better approximations of the model are used in the optimization problem to calculate the control input. The higher fidelity models require longer execution time; however, they yield a better approximation of the optimal control input.

3) Intuitively, the stability of the algorithm would improve if a terminal set constraint is imposed on the problem. However, such a constraint would also lead to larger processing time for the optimization problem; thus leading to more instances where the problem cannot be solved. The impact of this trade-off on performance is not entirely clear. Similarly, it is not a priori obvious if applying the control trajectory calculated at the previous time step, rather than applying zero control, if the problem cannot be solved will lead to better performance for all system parameters.

C. Numerical Examples

In this section, we illustrate that even for simple systems, the performance improvement by using the algorithm can be significant. We consider the process in the form (1) with matrices

\[
A = \begin{bmatrix} 1.1 & 0 \\ 0.1 & 0.5 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix}
\]

\[ \] (14)

and \( Q, R \) and \( R_w \) all being identity matrices. The control input \( u_1 \) is calculated first. Note that the process is already in the controllable canonical form with respect to \( u_1 \). We assume that the execution time available is uniformly distributed in the interval \([0, 1] \). The execution time can also be viewed as the fraction of the maximum possible processor time that is available at any time step. Finally, the time required to calculate both the inputs, if a \( 2 \times 2 \) matrix were multiplied by a 2-dimensional state vector is assumed to be equal to 0.6. We trace the evolution of the system over 10,000 time steps with identical noises and execution time availability patterns using our algorithm and the baseline algorithm described in Section III-A.2. Figure 1 shows the percentage improvement in cost over 10,000 steps for the same system for various exponents \( \alpha \) in the function \( f(p) = O(p^\alpha) \). As may be expected, the improvement becomes more pronounced as the execution time \( f(p) \) required to calculate \( p \) inputs increases faster. However, even for the linear case, a significant improvement in performance, approximately equal to 50\%, can be achieved.

![Fig. 1. Percentage improvement in cost by using the proposed algorithm for various functions \( f(p) \).](image-url)

We can also consider the receding horizon control implementation of the algorithm. We consider the constraint that each control value needs to be less than 0.1 in magnitude. We consider the same process as above, but with the noise \( w(k) \) being zero. We implemented a receding horizon control based algorithm using Matlab on a Windows XP machine. The optimization problem was solved using the \texttt{quadprog} function in Matlab, and the execution time noted using the functions \texttt{tic} and \texttt{toc}. The execution time available was assumed to be uniformly distributed in the interval \([0, 0.05] \) seconds. The simulation length was 50 time steps, and the achieved cost is plotted as a function of the horizon length for both the traditional and proposed algorithm in Figure 2. The plot shows some interesting features. As the horizon length is increased, the number of optimization variables increases. Thus, in general, we see a trend towards increasing costs, since with a higher frequency, control values are not calculated within the time available. In fact, the algorithm that ignores processor constraints is not able to stabilize the process for horizon length 8 and above. The proposed algorithm is able to stabilize the process for longer horizon lengths. For very small horizon lengths, there is sufficient time available for both algorithms to calculate both control inputs at every time step. Thus, the proposed algorithm performs worse since it calculates the control inputs one at
Horizon length

Cost achieved

Fig. 2. Achieved costs as a function of the horizon length in the RHC formulation.

ignoring processor constraints
Proposed algorithm

a time, rather than jointly as in the traditional algorithm. However, as the horizon increases and the complexity of the optimization problem to be solved increases, the performance of the proposed algorithm gradually becomes better than the traditional implementation.

IV. CONCLUSIONS

We proposed a simple control algorithm with anytime property. The algorithm was based on computing the control inputs sequentially, thus solving a sequence of optimization problems that refine the control input vector as more time becomes available. Alternatively, the algorithm can be looked as considering a sequence of control problems each with a better model of the process. For unconstrained linear systems, we provided analytic expressions for stability and performance gain with the algorithm. For constrained systems, we proposed a receding horizon control-based extension. Simple numerical examples illustrated the stability and performance gain with the proposed algorithm.

REFERENCES

[15] L. Schenato, “To hold or to zero control inputs with lossy links?”, IEEE Transactions on Automatic Control, Accepted.