Implementing Adaptive Predictive Control with the TMS320C50 DSP

Authors: J-M. Aymes

EFRIE, France

December 1995
SPRA311

TEXAS INSTRUMENTS

TRADEMARKS

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Implementing Adaptive Predictive Control with the TMS320C50 DSP

Abstract

This application report describes the implementation of the Texas Instruments (TI™) TMS320C50 digital signal process (DSP) and particularly the TMS320C50 DSP starter kit (DSK) as an advanced controller. The TMS320C50 effectively handles heavy computation loads required by adaptive predictive control methods. The design includes a supervisor to improve robustness to modeling errors.

To illustrate, an experiment is conducted to regulate the position of a marble on a rail. The marble on a rail is an unstable mechanical system in which dynamics change according to its state. Sophisticated methods of adaptive control are therefore necessary. These methods are composed mostly of a predictive controller and an algorithm to identify the parameters of the process to be controlled.

The identification is often realized by recursive least squares (RLS) methods that provide an estimated transfer function of the system in discrete time. For this experiment, the generalized predictive control (GPC) from D.W. Clarke was chosen as the control algorithm. The augmented UD identification (AUDI) from S. Niu was chosen as the identification algorithm. In adaptive control, the GPC is often employed because of its robustness properties and the AUDI because of its numerical properties.

This document was an entry in the 1995 DSP Solutions Challenge, an annual contest organized by TI to encourage students from around the world to find innovative ways to use DSPs. For more information on the TI DSP Solutions Challenge, see TI's World Wide Web site at www.ti.com.

Product Support on the World Wide Web

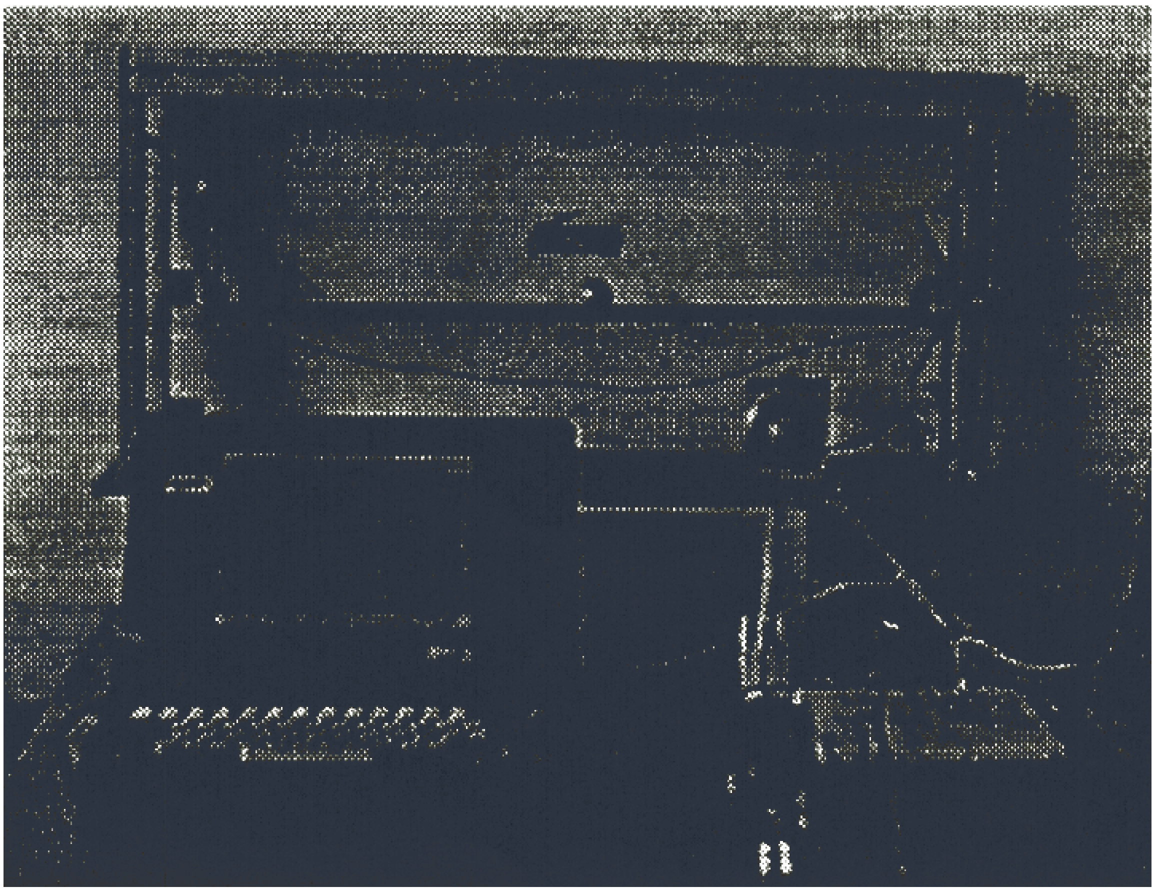
Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

The Experiment

Physical Description and Control Objective

The system is mostly composed of an iron marble moving on a reclining rail shaped like a gutter. One end of the rail is fixed to an axis and can freely turn around it. The other extremity is fixed to a spring pulled on its other side by a thread. The thread is rolled around a pulley in which rotation is controlled by an electric motor through a mechanical reducer. Figure 1 shows the system.

Figure 1. Experiment System

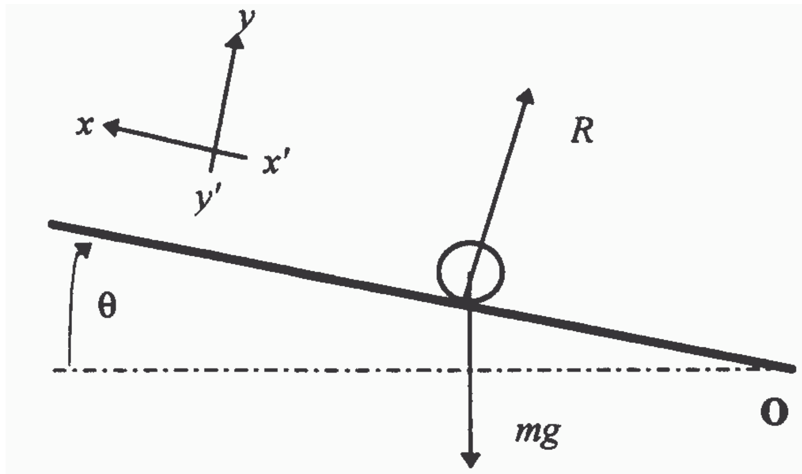


The control objective is to regulate the position of the marble in the middle of the rail. For example, the controller should be able to keep the marble in the middle of the rail, then make it go close to the left end and, finally, go back to the middle.

Modeling of the Mechanical System

To understand the difficulty of the control problem, it is necessary to write down the mechanical equations of the system. Two different parts can be considered. The first one concerns the way the marble is accelerated according to the rail angle. Practically, the marble is submitted to the forces shown in Figure 2

Figure 2. Marble Acceleration Model



The vector equation of mechanics for the marble gives:

$$m\vec{\gamma} = m\vec{g} + \vec{R} \qquad (1)$$

where:

m denotes the mass of the marble,

$\vec{\gamma}$ denotes its acceleration,

$m\vec{g}$ denotes its weight,

$\vec{R}$ denotes the reaction of the rail.

Here, no friction between the marble and the rail is assumed. If projected onto the x axis, this equation becomes:

$$m\ddot{x} = mg\cos\theta \qquad (2)$$

where:

x denotes the position of the marble,

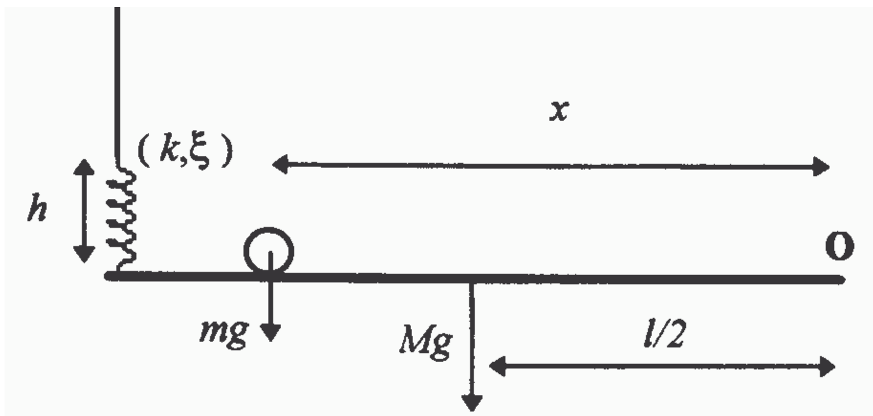
θ denotes the angular position of the rail.

So, for low values of the θ angle, the transfer function between θ and x is when using the Laplace transform:

$$\frac{X(s)}{\Theta(s)} = -\frac{g}{s^2} \tag{3}$$

Now follows the second part of the system concerning the way the angular position of the rail is set. To describe how the control of θ is achieved, it is necessary to consider the system marble + rail and to write the equation of the torque around the **O** point:

Figure 3. Rail Position Model



It yields to

$$I(x)\ddot{\theta} = -Mg\frac{1}{2} - mg\frac{x}{2} + (kh + \xi\,\dot{h})l \tag{4}$$

where:

l denotes the length of the rail,

M denotes its mass,

$I(x)$ denotes its inertia given by:

$$I(x) = \frac{1}{3}Ml^2 + mx^2 \tag{5}$$

k denotes the rigidity of the spring,

ξ denotes its dynamic damping ratio,

h denotes its elongation.

The elongation can be divided into three terms:

$$h = h' + h_0 - l\theta \tag{6}$$

where h_0 denotes the elongation needed to compensate the weight of the rail.

Then, the dynamic equation can be simplified:

$$I(x)\ddot{\theta} = -mg\frac{x}{2} + k(h' - l\theta)l + \xi(h' - l\theta)l \tag{7}$$

which is also:

$$I(x)\ddot{\theta} + \xi\, l^2\dot{\theta} + kl^2\theta = -mg\frac{x}{2} + \left(kh' + \xi\, \dot{h}'\right)l \tag{8}$$

Assuming slow motion, which means *I(x)* is nearly constant, and using the Laplace transform once more, the equation becomes:

$$\left[I(x)s^2 + \xi\, l^2 s + kl^2\right]\Theta(s) = -\frac{mg}{2}X(s) + l(k + \xi\, s)H'(s) \tag{9}$$

But:

$$\frac{X(s)}{\Theta(s)} = -\frac{g}{s^2} \tag{10}$$

So, if the true input of the system $u = \dot{h}'$ is used, the transfer function between θ and u is given by:

$$\frac{\Theta(s)}{U(s)} = \frac{l(k + \xi\, s)s}{I(x)s^4 + \xi\, l^2 s^3 + kl^2 s^2 - \frac{mg^2}{2}} \tag{11}$$

which finally gives the useful transfer function between the position of the marble and the control input:

$$\frac{X(s)}{U(s)} = \frac{1}{s}\frac{-gl(k + \xi\, s)}{I(x)s^4 + \xi\, l^2 s^3 + kl^2 s^2 - \frac{mg^2}{2}} \tag{12}$$

Assuming that ξ is negligible, the transfer function takes the form:

$$\frac{X(s)}{U(s)} = \frac{1}{s}\frac{K}{(s^2 + \alpha)(s^2 - \beta)} \tag{13}$$

where α and β are real and positive. So, the process includes:

- An integrator
- An oscillating mode
- An unstable mode

Moreover, these modes change as $I(x)$ changes with respect to the position of the marble. All these techniques made the process a very uneasy one to be controlled.

Measurement of the Marble Position

To control its position, the marble is measured by a potentiometric system. Because a resistive wire lies along the rail, the metallic marble acts as the cursor of the potentiometer. Figure 4 shows a section of the rail explaining the way the position is measured:

Figure 4. Rail Section

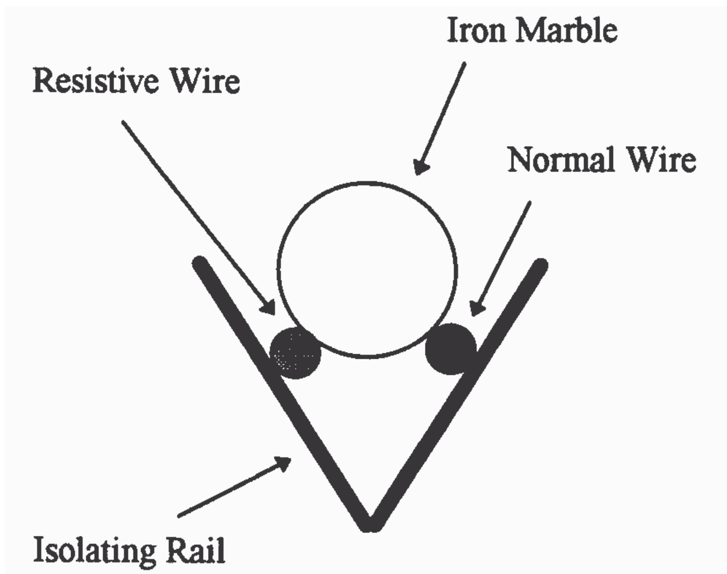
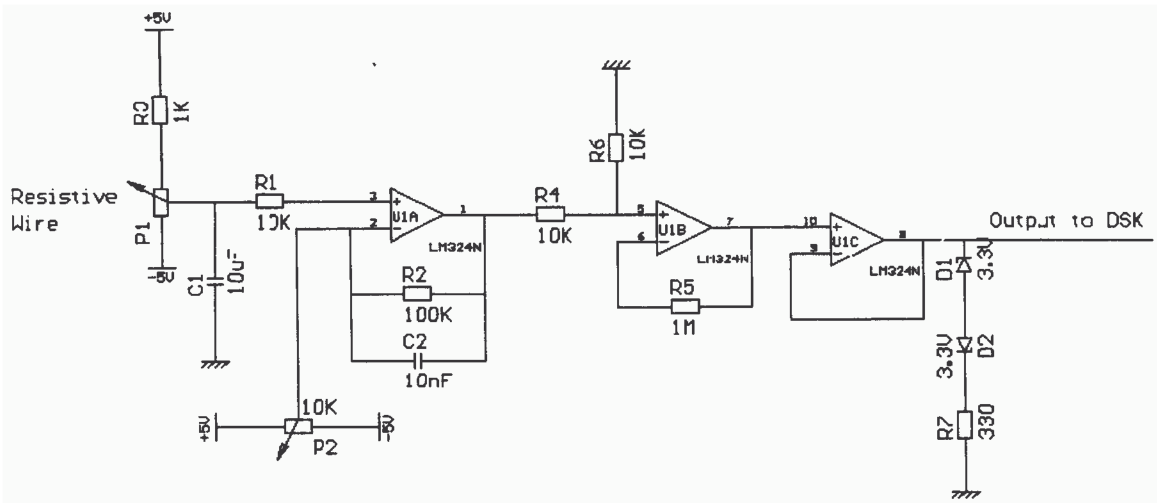


Figure 5 illustrates the electronic circuit that shapes the signal coming from the special potentiometer:

Figure 5. Electronic Circuit



Here, the capacitor C1 holds the input voltage in case the marble loses contact with the resistive wire. The potentiometer P2 helps to adjust the zero position. The first operational amplifier on the left amplifies and cuts off the high-frequency components of the signal. The second operational amplifier only amplifies. The last one adapts the output impedance. Note that the Zener diodes at the output limit the voltage in order to protect the A/N converter of the DSK board from high voltages.

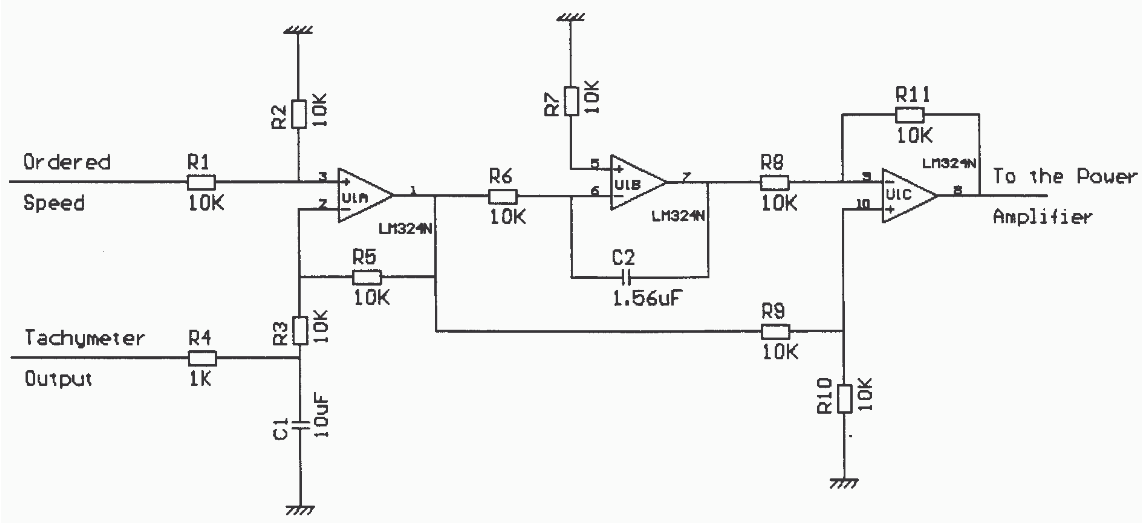
Primary Control of the Speed of the DC Electric Motor

The motor that controls the position of the rail has the following transfer function when it is unloaded:

$$T(s) = \frac{24}{s+30} \tag{14}$$

This includes the power amplifier. But when it is loaded and especially dynamically loaded, its behavior changes significantly in terms of response time and static gain. So, to avoid unpredictable results, it is necessary to add a controller to ensure good performances of the whole system. A very simple electronic proportional-integral (PI) regulator realizes it. The motor has an integrated tachymeter that provides the required value of speed after low pass filtering. Figure 6 shows the electronic diagram of the P1 controller:

Figure 6. PI Controller



The experiment shows that the transfer function of the closed loop remains regardless of the load:

$$T(s) = \frac{30}{s + 30} \tag{15}$$

GPC Control Algorithm

Main Controller

The control algorithm is the well-known GPC of D.W. Clarke, C. Mothadi, and P.S. Tuffs (Clarke et al., 1987). The process is supposed to be represented by an ARMAX model:

$$A\left(q^{-1}\right)y(t) = B\left(q^{-1}\right)u(t-d) + \frac{e(t)}{\Delta\left(q^{-1}\right)} \tag{16}$$

where:

y(t) denotes the output of the process,

$u(t)$ denotes its input,

$e(t)$ denotes an uncorrelated random noise,

$A\left(q^{-1}\right)$ and $B\left(q^{-1}\right)$ are polynomials of q^{-1}, the backward shift operator:

$A\left(q^{-1}\right) = 1 + a_1 q^{-1} + \ldots + a_{Na} q^{-Na}$ and

$$B\left(q^{-1}\right) = b_0 + b_1 q^{-1} + \ldots + b_{Nb} q^{-Nb} \tag{17}$$

$$\Delta\left(q^{-1}\right) = 1 - q^{-1}$$

The control objective is that the output *y(t)* follows the reference trajectory *r(t)* with dynamics specified by $P(q^{-1})$. So an auxiliary output is defined by:

$$\phi(t) = P\left(q^{-1}\right)y(t) \tag{18}$$

which is the output filtered by the desired dynamics of the closed loop.

The control increment minimizes the following cost function:

$$J = \sum_{i=hi}^{hp} \left[\hat{\phi}(t+i) - P(1)r(t)\right]^2 + \lambda \sum_{j=0}^{hc-1} \left[\Delta u(t+j)\right]^2 \tag{19}$$

under the constraint:

$$if\ \ k > 0, \Delta u(t+k) = 0 \tag{20}$$

where:

$\hat{\phi}(t+i) = E\left\{\frac{\phi(t+i)}{t}\right\}$ denotes the prediction of the auxiliary output at time $t+i$ according to t,

hi denotes the initial horizon,

hp denotes the prediction horizon,

hc denotes the control horizon,

λ denotes the weight on the inputs.

To simplify the algorithm, hc is assumed to be equal to 1. The predictors of $\phi(t)$ are given by:

$$\hat{\phi}(t+i) = F_i\left(q^{-1}\right)y(t) + E_i\left(q^{-1}\right)B\left(q^{-1}\right)\Delta u(t+i-d) \qquad (21)$$

where $E_i(q^{-1})$ and $F_i(q^{-1})$ are the polynomial solutions of the Diophantine equation:

$$P\left(q^{-1}\right) = A\left(q^{-1}\right)\Delta\left(q^{-1}\right)E_i\left(q^{-1}\right) + q^{-i}F_i\left(q^{-1}\right) \qquad (22)$$

$$\deg(E_i) = i-1$$

$$\deg(F_i) = \max\left[\deg(P), \deg(A)\right]$$

The following algorithm compute $E_i(q^{-1})$ and $F_i(q^{-1})$ recursively:

Initialization : set:

$$A'\left(q^{-1}\right) = A\left(q^{-1}\right)\Delta\left(q^{-1}\right)$$

$$E_1\left(q^{-1}\right) = p_0$$

$$F_1\left(q^{-1}\right) = q\left[P\left(q^{-1}\right) - p_0 A'\left(q^{-1}\right)\right]$$

where p_0 is the constant coefficient of the polynomial $P(q^{-1})$.

Recursion : Compute:

$$E_{i+1}\left(q^{-1}\right) = E_i\left(q^{-1}\right) + f_{i,0}q^{-i+1}$$

$$F_{i+1}\left(q^{-1}\right) = q\left[F_i\left(q^{-1}\right) - f_{i,0}A'\left(q^{-1}\right)\right]$$

where $f_{i,0}$ is the constant coefficient of the polynomial $F_i(q^{-1})$.

Now some auxiliary variables must be defined:

$$G_i(q^{-1}) = E_i(q^{-1})B(q^{-1}) = g_{i,0} + g_{i,1}q^{-1} + \ldots + g_{i,Nb+i-1}q^{-Nb-i+1}$$

$$h_i = g_{i,j-d} \tag{23}$$

$$H_i(q^1) = g_{i,j-d+1} + g_{i,j-d+2}\, q^{-1} + \ldots + g_{i,Nb+i-1}\, q^{-Nb-d+1}$$

So, the cost function can be rewritten:

$$J = \sum_{i=hi}^{hp} \left[F_i(q^{-1})y(t) + h_i\Delta u(t) + H_i(q^{-1})\Delta u(t-1) - P(1)r(t)\right]^2 + \lambda\,[\Delta u(t)]^2 \tag{24}$$

Its minimization implies:

$$\frac{\partial J}{\partial[\Delta u(t)]} = 0 = 2\sum_{i=hi}^{hp} h_i\left[F_i(q^{-1})y(t) + h_i\Delta u(t) + H_i(q^{-1})\Delta u(t-1) - P(1)r(t)\right] + 2\lambda\,\Delta u(t) \tag{25}$$

Finally, the control increment is given at time t by:

$$\Delta u(t) = \frac{P(1)\left(\sum_{i=hi}^{hp} h_i^2\right)r(t) - \left(\sum_{i=hi}^{hp} F_i(q^{-1})\right)y(t) - \left(\sum_{i=hi}^{hp} H_i(q^{-1})\right)\Delta u(t-1)}{\lambda + \sum_{i=hi}^{hp} h_i^2} \tag{26}$$

which can take the form used in the source code:

$$\Delta u(t) = \frac{Tr(t) - R(q^{-1})y(t) - S(q^{-1})\Delta u(t-1)}{\gamma} \tag{27}$$

Supervision

To achieve better robustness, the following rules have been included in the final design. They are essentially practical:

Rule 1

$$|\Delta u(t)| \le \Delta U_{\max} \tag{28}$$

The control increment is restricted to limit the divergence rate of the process in case of poor identification. So, the estimation algorithm is more likely to find a realistic model.

Rule 2

$$U_{\min} \leq u(t) = u(t-1) + \Delta u(t) \leq U_{\max} \tag{29}$$

The control increment itself is restricted to avoid the critical results of possible instabilities.

Rule 3

$$\left|\Delta u(t)\right| \leq \left|\frac{y(t) - r(t)}{b_0}\right| \tag{30}$$

In any case, the step of the control increment should exceed the one necessary to reach the ordered position in one step.

Rule 4

If $\Delta u(t)$ and $\Delta u(t-1)$ have opposite signs then

$$\left|\Delta u(t)\right| \leq \alpha \left|\Delta u(t-1)\right| \text{ with } 0 < \alpha < 1 \tag{31}$$

This condition avoids outputting a bang-bang control signal when the process identification performs poorly.

AUDI Identification Algorithm

AUDI as an RLS Algorithm

The AUDI algorithm was developed by S. Niu, D. Grant Fisher, and D. Xiao (Niu et al., 1992). This algorithm was largely inspired by the UD factorization algorithm of G.J. Bierman (Bierman, 1977). Because the AUDI is an RLS algorithm with a constant forgetting factor, it minimizes the following cost function:

$$J = \sum_{i=0}^{t} \lambda^{t-i} \left[y(i) - \underline{\theta}(i)^t \, \underline{\varphi}(i) \right]^2 \tag{32}$$

where:

λ denotes the forgetting factor ($0 < \lambda < 1$),

$y(t)$ denotes the output of the system,

$\underline{\varphi}(t)$ denotes the data vector such that:

$$\underline{\varphi}(t) = \left[-y(t-1) \cdots -y(t-n) \, u(t-d) \cdots u(t-d-m) \right]^t \tag{33}$$

$u(t)$ denotes the input of the system,

$\theta(t)$ denotes the vector of the system parameters such that:

$$\underline{\theta}(t) = \left[a_1 \cdots a_n \quad b_0 \cdots b_m \right]^t \tag{34}$$

The result of the minimization is the vector $\hat{\underline{\theta}}(t)$, which is the best estimate of the parameters. The corresponding estimated transfer function of the system is given by:

$$\hat{T}\left(q^{-1}\right) = q^{-d} \frac{\hat{b}_0 + \hat{b}_1 q^{-1} + \ldots + \hat{b}_m q^{-m}}{1 + \hat{a}_1 q^{-1} + \ldots + \hat{a}_n q^{-n}} \tag{35}$$

The following standard RLS algorithm obtains the vector $\hat{\underline{\theta}}(t)$ recursively:

At each sample time, compute:

$$\hat{\underline{\theta}}(t) = \hat{\underline{\theta}}(t-1) + \underline{K}(t) \left[y(t) - \underline{\varphi}(t)^t \, \hat{\underline{\theta}}(t-1) \right]$$

$$S(t) = \lambda + \underline{\varphi}(t)^t P(t-1) \underline{\varphi}(t)$$

$$P(t) = \frac{\left[P(t-1) - P(t-1)\underline{\varphi}(t)S^{-1}(t)\underline{\varphi}(t)^t P(t-1)\right]}{\lambda} \quad (36)$$

$$\underline{K}(t) = P(t)\underline{\varphi}(t)$$

where:

$\underline{K}(t)$ denotes the vectorial Kalman gain,

$P(t)$ denotes the covariance matrix of the parameter vector estimate.

The problem is that the formula updating $P(t)$ is potentially unstable because there is no guarantee that the matrix will always be positive definite. The main idea of the AUDI algorithm is to decompose the covariance matrix $P(t)$ into the $P(t) = U(t)D(t)U(t)^t$ form where $U(t)$ and $D(t)$ are a unit-upper-triangular matrix and a diagonal matrix respectively.

This decomposition guarantees positive definiteness of $P(t)$. In the algorithm, $U(t)$ and $D(t)$ are updated instead of $P(t)$. Another interesting feature of the algorithm rearranges and augments the data vector and the parameters vector. These new vectors are defined as:

$$\underline{\varphi}(t) = [-y(t-n) \quad u(t-d-n) \cdots -y(t-2) \quad u(t-d-1) \quad -y(t-1) \quad u(t-d) \quad -y(t)]^t$$

$$\underline{\theta_n}(t) = [a_n \quad b_n \quad \cdots \quad a_2 \quad b_2 \quad a_1 \quad b_1 \quad 1]^t \quad (37)$$

where n is the order of the estimate. The cost function the AUDI algorithm minimizes is:

$$J_n = \sum_{i=0}^{t} \lambda^{t-i} \left[\underline{\theta_n}(i)^t \underline{\varphi_n}(i)\right]^2 \quad (38)$$

The reason for choosing such vectors is explained by the forms taken by the matrix $U(t)$ and $D(t)$:

$$U(t) = \begin{bmatrix} 1 & \underline{\hat{\alpha}_0}(t-n) & & & & & & \\ & 1 & \underline{\hat{\theta}_1}(t-n+1) & & & & & \\ & & 1 & \underline{\hat{\alpha}_1}(t-n+1) & & \underline{\hat{\alpha}_{n-1}}(t-1) & \underline{\hat{\theta}_n}(t) \\ & & & 1 & & & \\ & & & & \ddots & & \\ & 0 & & & & 1 & \\ & & & & & & 1 \end{bmatrix}$$

$$D(t) = \{diag[J_0(t-n)\, L_0(t-n) \cdots L_{n-1}(t-1)\, J_n(t)]\}^{-1} \qquad (39)$$

where, particularly:

$\underline{\hat{\theta}_i}(t)$ denotes the estimate of the parameters of the ith order model,

$J_i(t)$ denotes the value of the cost function for the ith order model.

This algorithm provides simultaneous estimates of the parameters for all model orders from 1 to n with a computational load equivalent to nth order RLS.

Stepwise Procedure for the AUDI Algorithm

Initialization: At $t=0$, set:

$$P(0) = U(0)D(0)U(0)^t = \sigma^2 I$$

where σ is a large integer and I is the identity matrix.

Step 1: Construct the data vector $\underline{\varphi}(t)$ and compute:

$$\underline{f} = U(t-1)^t \varphi(t)$$
$$\underline{g} = D(t-1)\underline{f}$$

Set $\beta_0 = \lambda$

Step 2: For $j = 1, \cdots, d,$ go through steps 3-5 ($d=2n$ is the dimension of $U(t)$ and $D(t)$).

Step 3: Compute :

$$\beta_j = \beta_{j-1} + f_j\, g_j$$
$$D(t)_{j,j} = \frac{\beta_{j-1} D(t-1)_{j,j}}{\beta_j \lambda}$$
$$v_j = g_j$$
$$\mu_j = -\frac{f_j}{\beta_{j-1}}$$

Step 4: If $j>1$, for $i=1,\ldots,j-1$, go through step 5.

Step 5: Compute:

$$U(t)_{i,j} = U(t-1)_{i,j} + v_i \mu_j$$
$$v_i = v_i + U(t-1)_{i,j} v_j$$

At the end of the computation, the parameter vector estimate is given by the last column of the $U(t)$ matrix.

C Source Code of the Whole Controller

`#include <dsk.h>`	includes basic function of the DSK board (described in Appendix)
`#define min(a,b) ((a)<(b)) ? (a):(b)`	Some basic functions …
`#define abs(x) ((x)>0)?(x):-(x)`	
`#define N 5`	degree of the estimated transfer function
`#define d 5`	delay of the process
`float A[N+l], B[N];`	polynomials composing the transfer function
`#define Nd 2*N+1`	dimension of U and D matrix
`#define ID_forget 0.95`	forgetting factor of the AUDI
`float U[Nd][Nd];`	matrix U of the AUDI
`float D[Nd];`	diagonal elements of matrix D
`float Phi[Nd];`	data vector of the AUDI
`#define Umax 10000.0`	maximum control input
`#define Umin –10000.0`	minimum control input
`#define DUmax 1000.0`	maximum input step
`#define LAMBDA 0.001`	weight on the input step of the GPC
`#define Np 2`	number of elements of polynomial P
`float P[Np]={ 1.0, -0.8};`	P itself
`#define Pl 0.2`	sum of the elements of P, P(1)
`#define hp 15`	prediction horizon
`float E[hp];`	polynomial Ei of the prediction of ϕ(t+i)
`float F[N+1];`	polynomial Fi of the prediction of ϕ (t+I)
`float R[N+l];`	polynomial R of the final form of the step input
`float S[N+d-2];`	polynomial S of the final form of the step input
`float T, gamma;`	T and γ
`float y0;`	ordered position of the marble
`float y[N+1];`	array containing past value of y(t) filtered by R
`float delta_u[N+d-2];`	array containing past value of Δu (t) filtered by S
`float Sample_In_10Hz=0.0, Sample_Out_10Hz=0.0;`	Input and Output of the 10 Hz interrupt procedure
`int Ok_10Hz=0;`	Flag indicating whether or not 10 hz procedure has to be entered
`void Array_Shift_Left(float *tab, int n, float x)`	Procedure to shift an array to the left

```
void Array_Shift_Left(float *tab, int n, float x)
{
    int i;
    for (i=1; i<n; i++) tab[i-1]=tab[i];
    tab[n-1] =x
}
```

`void Array_Shift_Right(float *tab, int n, float x)`	Procedure to shift an array to the right

```
void Array_Shift_Right(float *tab, int n, float x)
{
    int i;
```

```
    for (i=n-1; i>0; i--) tab[i]=tab[i-1];
    tab[0]=x;
}

float Array_Sum(float *a, float *b, int n)
{
    int i;
    float s = 0,0;
    for (i=0; i<n; i++) s+=a[i]*b[i];
    return s;
}
```

Procedure to compute scalar product of two arrays (used to filter)

```
void Array_Zero(float *tab, int n)
{
    int i;

    for (i=0; i<n; i++) tab[i]=0.0;
}
```

Set all elements of an array to zero

```
void Array_Product(float *a, float *b, int na, int nb, float *c)
{
    int i,j;

    tab_zero(c, na+nb-1);
    for (i=0; i<na; i++)
    {
        float temp=a[i];

        for j=0; j<nb; j++) c[i+j] +=temp*b[j];
    }
}
```

Multiply two arrays

```
void Array_Accumulate(float *a, float *b, int n, float x)
{
    int i;
    for (i=0; i<n; i++) a[i] += x*b[i];
}
```

Add one array to another according to a specified weight

```
void Init_AUDI(void)
{
    int i,j;

    Array_Zero(Phi, Nd);
    for (i=0; i<Nd; i++)
    {
        D[i]=1e10;
        for (j=0;j<Nd;j++) U[i][j]=(float)(i==j);

    }
    U[Nd-2][Nd-1]=1.0;
}
```

Initialization procedure of the AUDI algorithm

Set φ(t) to zero

all diagonal elements of D are set to 1e10
only diagonal elements of U are set to 0
(the others are zeroed) apart from the one of the last column which is actually b_0

```
void AUDI(void)                                          // The AUDI algorithm
{
    int i,j;
    float f[Nd], g[Nd], v[Nd], mu, fact, bo=ID_oubli, ff=0.0;

    for (i=0; i<Nd; i++)
    {
        f[i]=0.0;
        for (j=0; j<Nd; j++) f[i] +=U[j][i]*Phi[j];     // Compute f as described in step 1
    }
    for (i=0; i<Nd; i++) g[i]=D[i]*f[i];                 // Compute g
    fact=f[Nd-1]*f[Nd-1]*D[Nd-1];                        // Compute a measurement of how new is the
    if (fact<l.3*(l-ID_oubli)) return;                   // information in φ(t) and eventually abort the
                                                         // updating
    if (fact>0.5) fact=0.5;                              // Limit the updating
    for (i=0; i<Nd; i++) ff +=f[i]*g[i];                 // Allow to update only in the direction where
                                                         // there is new information

    if (ff>le-20) for (i=0; i<Nd; i++) g[i]*=(1-fact/ff);

    for (i=0; i<Nd; i++)                                 // Step 2 of the algorithm
    {
        float bn;

        bn=bo+f[i]*g[i];                                 // Step 3
        D[i] *=bo/bn/ID_oubli;
        v[i]=g[i];
        mu=-f[i]/bo;
        bo=bn;
        if (i>0)                                         // Step 4
            for (j=0; j<i; j++)
            {
                 float a;

                 a=U[j][i];                              // Step 5
                 U[j][i]=a+v[j]*mu;
                 v[j]+=a*v[i];
            }
    }
    for (i=0; i<N; i++)
    {
        B[i]=U[Nd-2*(i+l)][Nd-l];                        // Update the polynomials A and B according to
        A[i+1]=U[Nd-2*(i+1)-1][Nd-1];                    // the new last column of matrix U
    }
    A[0]=1.0;
}

void Init_GPC(void)                                      // Initialization of the GPC algorithm
{
    y0=0.0;                                              // Set the ordered position of the marble, the past
    Array_Zero(y, N+1);                                  // values of y(t) and the past values of Δu(t) to
    Array_Zero(delta_u, N+d-2);                          // zero
```

```
}

void GPC(void)                                          The GPC algorithm
{
    int i,j;
    float h,sum_h2=0.0;

    Array_Zero(R, N+1);                                 Set R and S to zero
    Array_Zero(S, N+d-2);
    for (i=0; i<hp; i++)
    {
        if(i==0)                                        Compute the polynomials Ei and Fi
        {
            E[0]=P[0]/A[0];                             Initialization
            for (j=0; j<N; j++)
            {
                 F[j]=-E[0]*(A[j+1]-A[j]);
                 if(j<=Np) F[j]+=P[j];
            }
            F[N]=E[0]*A[N];
        }
        else
        {
            E[i]=F[0]/A[0];                             Recursion

            for (j=0; j<N; j++) F[j]=F[j+l]-E[i]*(A[j+1]-A[j]);
            F[N]=E[i]*A[N];
        }
        if(i>=d-1)                                      Note that this condition implies hi = d
        {
            Array_Product(B,E,N,i+1,G);                    Compute Gi
            h=G[i+1-d];                                    and hi
            Array_Accumulate(S,&(G[i+2-d]),N+d-2,h);       Add Hi to S
            Array_Accumulate(R,F,N+1,h);                   Add Fi to R
            sum_h2 +=h*h;                                  Compute the sum of hi^2
        }
    }
    T=P1*sum_h2                                         Compute T
    gamma=LAMBDA + sum_h2;                              and y
}
interrupt void AIC_Interrupt(void)                      Interrupt procedure called every 1/5000 sec.
{
    static int count=0, Out =0;
    static float In=0.0;

    In=0.998*In+0.002*(float)AIC_Load();                Low pass filtering of the input
    if (++count==500)
    {
        count=0;                                        Every 1/10 sec. the procedure set the input and
        Sample_In_10Hz=In;                              the output for the 10 Hz procedure
        Out=(int)Sample_Out_10Hz;
```

```
        Ok_10Hz=l;
    }
    AIC_Write(Out);
}

void IT_10Hz(void)                                      Procedure called every 1/10 sec.
{
    static int count=0;
    float du, du_max, b1,b2;

    Ok_10Hz=0;
    if (count++> 100)
    {
        float next_y0;

        count=0;                                        Every 10 sec. the ordered position changes:

        if (y0==0.0) next_y0=10000.0;                   middle to left corner
        if(y0==10000.0) next_y0= -10000.0;              left corner to right corner
        if(y0== -10000.0) next_y0=0.0;                  back to the middle
        y0=next_y0;                                     and so on...
    }
    du_max=min(DUmax,abs((entree-position)/B[0]));      Set the maximum step in control input
                                                        according to the model
    Array_Shift_Right(y, Np+N, Sample_In_10Hz);         Store the new position

    du=(T*y0-Array_Sum(R,y,N+1)-Array_Sum(S,delta_u,N+d-2))/gamma;    Compute the new
                                                                      step of the control
                                                                      increment
    b1=min(du_max, Umax - Sample_Out_10Hz);             Look for a change of direction of the control
    b2=min(du_max, Sample_Out_10Hz - Umin);             input
    if (delta_u[0]>0.0) b2=min(b2, 0.1*delta_u[0]);     If there is one, limit the step to one tenth of
    if (delta_u[0] <0.0) b1=min(bl, -0.l*delta_u[0]);   the former step
    if(du>bl) du=b1;
    if(du<-b2)du= -b2;
    Array_Shift_Right(delta_u, N+d-2, du);              Store the new step
    Array_Shift_Left(Phi, Nd, - Sample_In_10Hz);        update φ(t) with the new position
    AUDI();                                             Run the AUDI procedure
    GPC();                                              Then run the GPC one
    Sample_Out_10Hz +=du;                               Set the control input
    Array_Shift_Left(Phi, Nd, Sample_Out_10Hz);         update φ (t) with the new control input
}

void main(void)                                         The main procedure
{
    Init_Hardware();                                    Initialization of the DSK board
    /*Fc=2KHz   Fe=5KHz*/
    AIC_Setup(31,31,32,32,GO|SYNCH);                    Set Sampling rate to 5 Khz and Cut-off
                                                        frequency to 2 KHz
    Init_AUDI();                                        Initialization of the AUDI algorithm
    Init_GPC();                                         Initialization of the GPC algorithm
    Enable_Interrupt();                                 Let's go !!!
```

```
    while(l)
    {
        if(Ok_l0Hz) IT_10Hz();                  Every 1/10 Hz enter the 10 Hz procedure
    }
}
```

Summary

As expected, the adaptive predictive controller performed very well. The stability margin seemed to be large. A new experiment based on the inverted pendulum has just shown that higher sampling rate can be achieved by the algorithm running in the Texas Instruments TMS320C50 DSP.

For example, a simultaneous sampling frequency of 100 Hz and a prediction horizon of 40 are no longer a problem for the TMS320C50 DSP. Therefore, the largest part of the most sophisticated methods of control can reach the status of real-time methods.

Acknowledgments

The support of R. Barthes, C. Berto, P. Duvernois, and 0. Mendiara is gratefully acknowledged.

References

The GPC controller:

D.W. Clarke, C. Mothadi, P.S. Tuffs (1987): "Generalized Predictive Control Part 1 & 2," *Automatica*, Vol.23, no.2, pp 137-160.

For more details:

G.Favier (1987): "Self-Tuning Long-Range Predictive Controllers," *IFAC 10th Triennal World Congress*, Munich, pp 83-90.

I.D. Landau (1993): "Evolution of Adaptive Control," *Journal of Dynamic Systems, Measurement and Control*, Vol.115, pp 381-391.

The AUDI algorithm:

S. Niu, D., Grant Fisher, D. Xiao (1992): "An Augmented U') Identification Algorithm," *International Journal of Control*, Vol.56, no.1, pp 1193-211.

For more details:

D.W. Clarke (1995): "Adaptive Predictive Control," *ACASP 5th Symposium*, Budapest, pp 43-54.

G.H. Golub, C.F. Van Loan (1989): In : *Matrix Computations*, Second Edition, The Johns Hopkins University Press Ltd, Chap. 4, pp 137-139.

Appendix A.

The following three files are required to compile the C source code and make it work with the TMS320C50 DSK.

File: “DSK.LNK”

```
MEMORY
{
    PAGE 0:
        IT:     origin=800h, length=  40h
        PROG:   origin=980h, length=1680h
    PAGE 1:
        DATA:   origin=2000h, length=0C00h
}
SECTIONS
{
    vectors:   >IT
    text:      >PROG
    cinit:     >PROG
    switch:    >DATA
    const:     >DATA
    stack:     >DATA
    sysmem:    >DATA
    data:      >DATA
    bss:       >DATA
}
```

File: “DSK.ASM"

```
          .title “DSK_STARTER
          .mmregs

          .sect     "vectors"
          .global   _c_inf0, _AIC_Interrupt

RESET:    B_c_int0                        ;RESET
INTl:     RETE                            ;Int 1
          NOP
INT2:     RETE                            ;lnt 2
          NOP
INT3:     RETE                            ;Int 3
          NOP
TINT      RETE                            ;TIMER
          NOP
RINT:     B         _AIC_Interrupt        ;Serial port receive
XINT:     RETE                            ;Serial transmit
          NOP
TRNT:     RETE                            ;TDM receive
          NOP
TXNT:     RETE                            ;TDM transmit
```

```
         NOP
INT4:    RETE                           ;lnt4
         NOP

         .text.
         global_AIC_Init
         global_AIC_Load
         global_AIC_Write
         global_TA,_RA,_TB,_RB,_AIC_CTR

_AIC_Init:
         SETC     INTM
         LDP      #0
         OPL      #0830h,PMST
         ZAC
         SAMM     CWSR
         SAMM     PDWSR
         SPLK     #0022h,IMR
         CALL     AICINIT
         LDP      #0
         SPLK     #0012h,IMR
         CLRC     OVM
         SPM      0
         MAR      *,ARl
         RET

AIC_Load:
         LAMM     DRR
         AND      #0fffch
         RET

_AIC_Write:
         SAR      AR0,*+
         SAR      AR1,*
         LAR      AR0,*+,AR2
         LARK     AR2,-2
         MAR      *0+
         LACC     *,AR1
         AND      #0FFFCh
         SAMM     DXR
         RETD
         SBRK     2
         LAR      ARO,*

AICINIT: LDP      #0
         SETC     SXM
         SPLK     #0020h,TCR
         SPLK     #0001h,PRD
         MAR      *,AR3
         LACC     #0008h                ;Non continuous mode
         SACL     SPC                   ;FSX as input
         LACC     #00C8h                ;l6 bit words
```

```
            SACL      SPC
            LACC      #080h                     ;Pulse AIC reset by seting it low
            SACH      DXR
            SACL      GREG
            LAR       AR3,#0FFFFh
            RPT       #10000                    ;and taking it high after 10000 cycles
            LACC      *,0,AR3                   ;(.5ms at 50ns)
            SACH      GREG
            ;__________________
            LDP       #_TA
            LACC      _TA,9                     ;Initialized TA and RA register
            LDP       #_RA
            ADD       _RA,2
            CALL      AIC_2ND
            ;__________________
            LDP       #_TB
            LACC      _TB,9                     ;Initialized TB and RB register
            LDP       #_RB
            ADD       _RB,2
            ADD       #02h
            CALL      _AIC_2ND
            ;__________________
            LDP       #_A_AIC_CTR
            LACC      _AIC_CTR,2                ;Initialized control register
            ADD       #03h
            CALL      AIC_2ND
            RET

AIC_2ND:
            LDP       #0
            SACH      DXR
            CLRC      INTM
            IDLE
            ADD       6h,15
;0000 0000 0000 0011 XXXX XXXX XXXX XXXX b
            SACH      DXR
            IDLE
            SACL      DXR
            IDLE
            ZAC
            SACL      DXR                       ;make sure the word got sent
            IDLE
            SETC      INTM
            RET

            .text

            .global_Init_Hardware
            .global_Enable_Interrupt
            _global_Disable_Interrupt

_Init_Hardware
```

```
          SETC      INTM
          LDP       #0
          OPL       #0830h,PMST
          ZAC
          SAMM      CWSR
          SAMM      PDWSR
          SETC      SXM
          CLRC      OVM
          SPM       #0
          RET

_Enable_Interrupt:
          CLRC      INTM
          RET

_Disable_Interrupt:
          SETC      INTM
          RET

          .end
```

File: “DSK.H"

```
/*__________________________________________________   */
/* MCLK=10 MHz                                          */
/* SCLK=MCLK/4=2.5 MHz                                  */
/* SCF=MCLK/2/TA                                        */
/* Fout=3.5*SCF/288-6l/TA                               */
/* Fs=MCLK/2/TA/TB                                      */
/* Example:                                             */
/* Fs=20 KHz                                            */
/* TA=RA=7       TB=RB=36                               */
/*__________________________________________________   */

/*Bit definition of the control register in the AIC    */

# define BANDPASS     1
# define LOOPBACK     2
# define AUXIN        4
# define SYNCH        8
# define G0           16
# define Gl           32
# define SINX_X       128

extern int TA=24,RA=24,TB=18,RB=18;
extern int AIC_CTR=GO|SYNCH;
extern void AIC_Init(void);

void AIC_Setup(int new_ta, int new_ra, int new_tb, int new_rb, int new_aic_ctr)
{
            TA=new_ta;
            RA=new_ra;
```

```
            TB=new_tb;
            RB=new_rb;
            AIC_CTR=new_aic_ctr;
            AIC_Init();
}

extern void AIC_Write(int);
extern int AIC_Load(void);
extern void Enable_Interrupt(void);
extern void Disable_Interrupt(void);
extern void Init_Hard(void);
```