

Disclaimer: This document was part of the DSP Solution Challenge 1995 European Team Papers. It may have been written by someone whose native language is not English. TI assumes no liability for the quality of writing and/or the accuracy of the information contained herein.

Implementing Adaptive Predictive Control with the TMS320C50 DSP

Authors: J-M. Aymes

EFRIE, France
December 1995
SPRA311



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

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

Abstract	7
Product Support on the World Wide Web	8
The Experiment.....	9
Physical Description and Control Objective.....	9
Modeling of the Mechanical System	10
Measurement of the Marble Position	13
Primary Control of the Speed of the DC Electric Motor	14
GPC Control Algorithm	16
Main Controller	16
Supervision	18
Rule 1	18
Rule 2	19
Rule 3	19
Rule 4	19
AUDI Identification Algorithm	20
AUDI as an RLS Algorithm.....	20
Stepwise Procedure for the AUDI Algorithm.....	22
C Source Code of the Whole Controller	24
Summary.....	29
Acknowledgments.....	29
References	30
Appendix A.....	31
File: "DSK.LNK"	31
File: "DSK.ASM"	31
File: "DSK.H"	34

Figures

Figure 1.	Experiment System	9
Figure 2.	Marble Acceleration Model.....	10
Figure 3.	Rail Position Model.....	11
Figure 4.	Rail Section	13
Figure 5.	Electronic Circuit	14
Figure 6.	PI Controller.....	15

Implementing Adaptive Predictive Control with the TMS320C50 DSP

Abstract

This application report describes the implementation of the Texas Instruments (TITM) 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

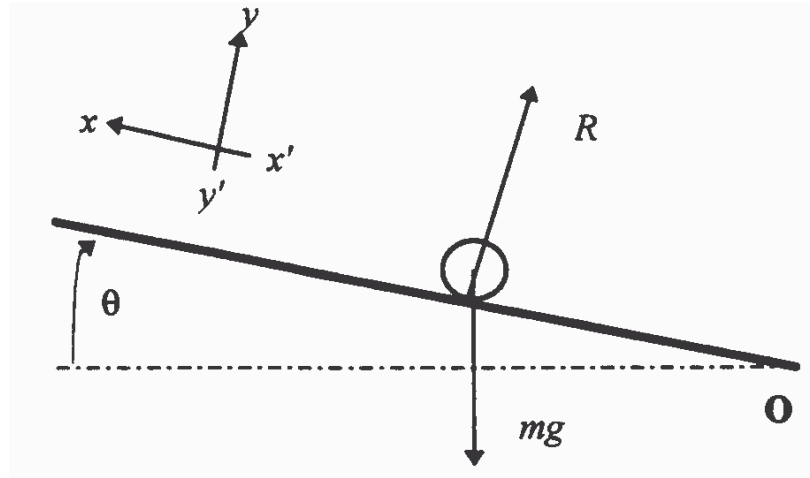


The control objective is to regulate the position of the marble in the middle of the rail. For example, the controller should be able to keep the marble in the middle of the rail, then make it go close to the left end and, finally, go back to the middle.

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:

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

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:

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

where:

x denotes the position of the marble,

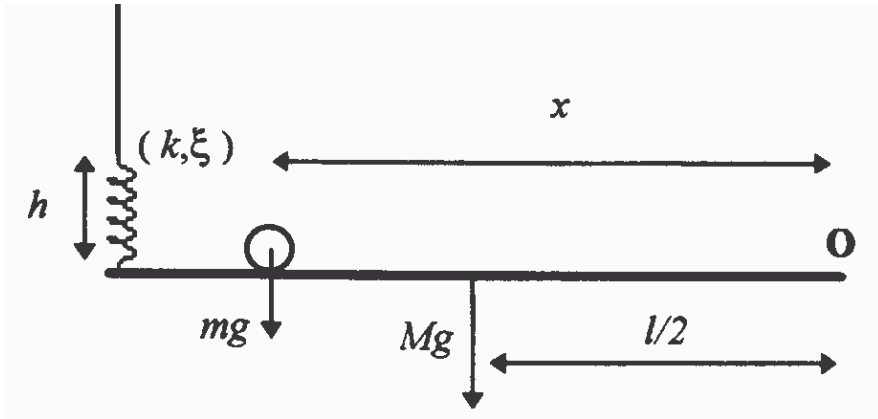
θ denotes the angular position of the rail.

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

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

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

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

where:

l denotes the length of the rail,

M denotes its mass,

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

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

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:

$$h = h' + h_0 - l\theta \quad (6)$$

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

Then, the dynamic equation can be simplified:

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

which is also:

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

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

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

But:

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

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

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

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

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

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

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

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

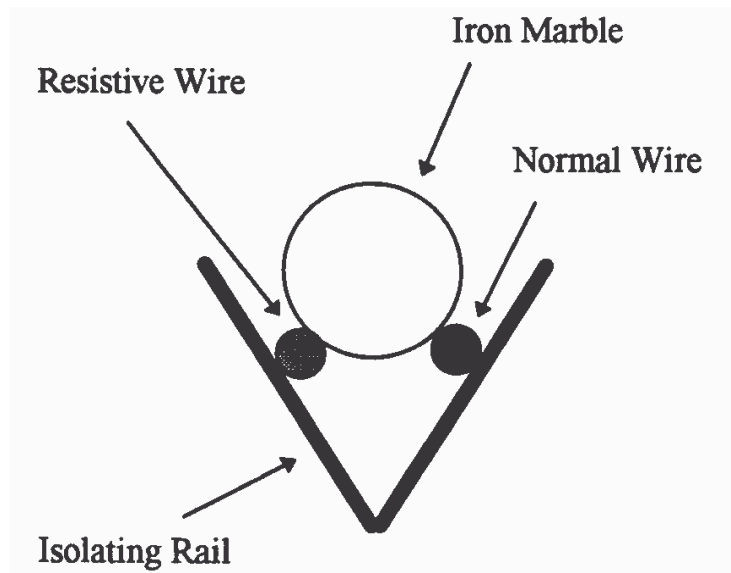
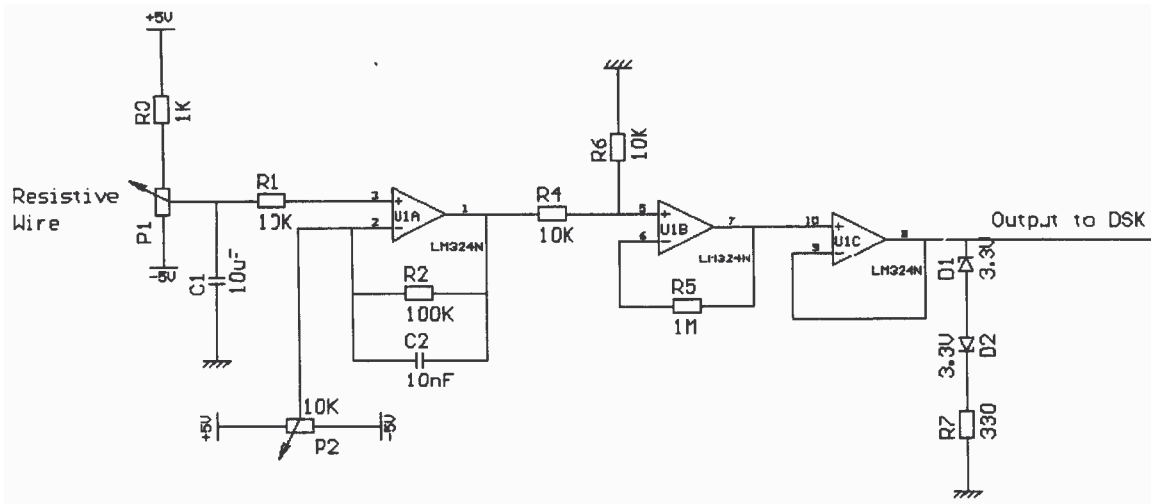


Figure 5 illustrates the electronic circuit that shapes the signal coming from the special potentiometer:

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.

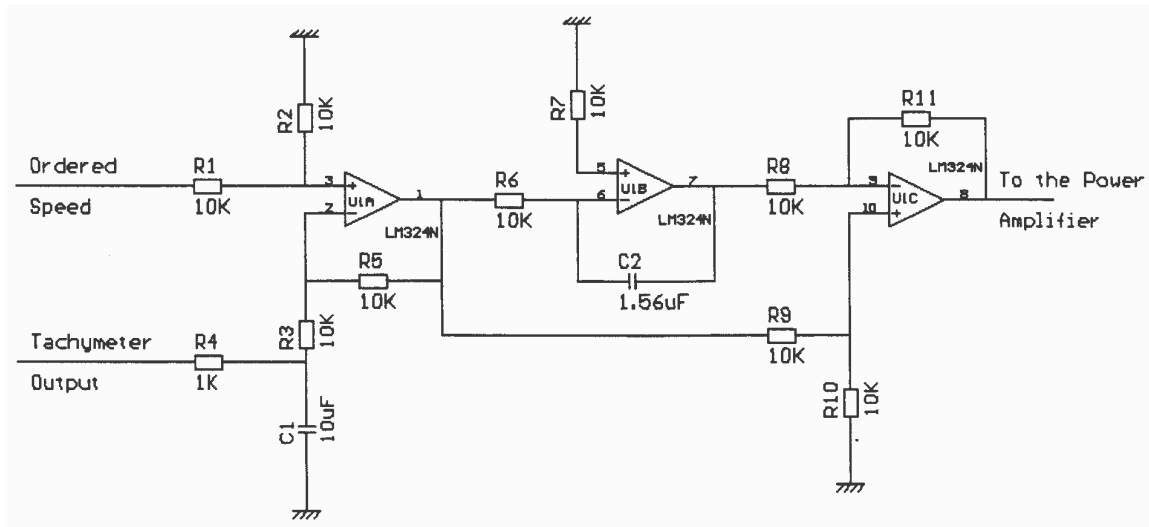
Primary Control of the Speed of the DC Electric Motor

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

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

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:

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

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:

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

where:

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

$u(t)$ denotes its input,

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

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

$$\begin{aligned} A(q^{-1}) &= 1 + a_1 q^{-1} + \dots + a_{Na} q^{-Na} \quad \text{and} \\ B(q^{-1}) &= b_0 + b_1 q^{-1} + \dots + b_{Nb} q^{-Nb} \\ \Delta(q^{-1}) &= 1 - q^{-1} \end{aligned} \quad (17)$$

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:

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

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

The control increment minimizes the following cost function:

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

under the constraint:

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

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:

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

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

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

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

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

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

Initialization : set:

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

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

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

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

Recursion : Compute:

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

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

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

Now some auxiliary variables must be defined:

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

$$h_i = g_{i,j-d} \quad (23)$$

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

So, the cost function can be rewritten:

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

Its minimization implies:

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

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

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

which can take the form used in the source code:

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

Supervision

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

Rule 1

$$|\Delta u(t)| \leq \Delta U_{\max} \quad (28)$$

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

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

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

Rule 3

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

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

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

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:

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

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:

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

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

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

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

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:

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

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

At each sample time, compute:

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

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

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

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

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:

$$J_n = \sum_{i=0}^t \lambda^{-i} [\underline{\theta}_n(i)^T \underline{\varphi}_n(i)]^2 \quad (38)$$

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

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

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

where, particularly:

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

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

This algorithm provides simultaneous estimates of the parameters for all model orders from 1 to n with a computational load equivalent to n th 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:

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

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

$$\text{Set } \beta_0 = \lambda$$

Step 2: For $j=1, \dots, d$, 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 $j > I$, for $i=1, \dots, j-I$, 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 <dsb.h>	includes basic function of the DSK board (described in Appendix) Some basic functions ...
#define min(a,b) ((a)<(b)) ? (a):(b)	
#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+1], 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 $\phi(t+i)$
float F[N+1];	polynomial Fi of the prediction of $\phi(t+l)$
float R[N+1];	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 $\Delta u(t)$ 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
{	
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
{	
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;
}

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

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

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];
    }
}

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

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;
}

```

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

Set all elements of an array to zero

Multiply two arrays

Add one array to another according to a specified weight

Initialization procedure of the AUDI algorithm

Set $\phi(t)$ 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)
{
    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<1.3*(1-ID_oubli)) return;                  information in  $\phi(t)$  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>1e-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+1)][Nd-1];
        A[i+1]=U[Nd-2*(i+1)-1][Nd-1];
    }
    A[0]=1.0;
}

void Init_GPC(void)
{
    y0=0.0;
    Array_Zero(y, N+1);
    Array_Zero(delta_u, N+d-2);
}

```

The AUDI algorithm

Step 2 of the algorithm

Step 3

Step 4

Step 5

Update the polynomials A and B according to the new last column of matrix U

Initialization of the GPC algorithm

Set the ordered position of the marble, the past values of $y(t)$ and the past values of $\Delta u(t)$ to zero



```

}

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

    Array_Zero(R, N+1);
    Array_Zero(S, N+d-2);
    for (i=0; i<hp; i++)
    {
        if(i==0)
        {
            E[0]=P[0]/A[0];
            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];

            for (j=0; j<N; j++) F[j]=F[j+1]-E[i]*(A[j+1]-A[j]);
            F[N]=E[i]*A[N];
        }
        if(i>=d-1)
        {
            Array_Product(B,E,N,i+1,G);
            h=G[i+1-d];
            Array_Accumulate(S,&(G[i+2-d]),N+d-2,h);
            Array_Accumulate(R,F,N+1,h);
            sum_h2 +=h*h;
        }
    }
    T=P1*sum_h2
    gamma=LAMBDA + sum_h2;
}

interrupt void AIC_Interrupt(void)
{
    static int count=0, Out =0;
    static float In=0.0;

    In=0.998*In+0.002*(float)AIC_Load();
    if (++count==500)
    {
        count=0;
        Sample_In_10Hz=In;
        Out=(int)Sample_Out_10Hz;
    }
}

```

The GPC algorithm

Set R and S to zero

Compute the polynomials E_i and F_i

Initialization

Recursion

Note that this condition implies $h_i = d$

Compute G_i and h_i

Add H_i to S

Add F_i to R

Compute the sum of h_i^2

Compute T and γ

Interrupt procedure called every 1/5000 sec.

Low pass filtering of the input

Every 1/10 sec. the procedure set the input and the output for the 10 Hz procedure

```

        Ok_10Hz=1;
    }
    AIC_Write(Out);
}

void IT_10Hz(void)
{
    static int count=0;
    float du, du_max, b1,b2;

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

        count=0;

        if (y0==0.0) next_y0=10000.0;
        if(y0==10000.0) next_y0= -10000.0;
        if(y0== -10000.0) next_y0=0.0;
        y0=next_y0;

        du_max=min(DUmax,abs((entree-position)/B[0]));
        Array_Shift_Right(y, Np+N, Sample_In_10Hz);
        du=(T*y0-Array_Sum(R,y,N+1)-Array_Sum(S,delta_u,N+d-2))/gamma;
        b1=min(du_max, Umax - Sample_Out_10Hz);
        b2=min(du_max, Sample_Out_10Hz - Umin);
        if (delta_u[0]>0.0) b2=min(b2, 0.1*delta_u[0]);
        if (delta_u[0] <0.0) b1=min(b1, -0.1*delta_u[0]);
        if(du>b1) du=b1;
        if(du<-b2)du= -b2;
        Array_Shift_Right(delta_u, N+d-2, du);
        Array_Shift_Left(Phi, Nd, - Sample_In_10Hz);
        AUDI();
        GPC();
        Sample_Out_10Hz +=du;
        Array_Shift_Left(Phi, Nd, Sample_Out_10Hz);
    }
}

void main(void)
{
    Init_Hardware();
    /*Fc=2KHz    Fe=5KHz*/
    AIC_Setup(31,31,32,32,GO|SYNCH);

    Init_AUDI();
    Init_GPC();
    Enable_Interrupt();
}

```

Procedure called every 1/10 sec.

Every 10 sec. the ordered position changes:

middle to left corner
left corner to right corner
back to the middle
and so on...

Set the maximum step in control input according to the model

Store the new position

Compute the new step of the control increment

Look for a change of direction of the control input

If there is one, limit the step to one tenth of the former step

Store the new step

update $\phi(t)$ with the new position

Run the AUDI procedure

Then run the GPC one

Set the control input

update $\phi(t)$ with the new control input

The main procedure

Initialization of the DSK board

Set Sampling rate to 5 Khz and Cut-off frequency to 2 KHz

Initialization of the AUDI algorithm

Initialization of the GPC algorithm

Let's go !!!

```
while(1)
{
    if(Ok_10Hz) 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 O. 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 Triennial 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_int0, _AIC_Interrupt

RESET:     B_c_int0                ;RESET
INT1:      RETE                    ;Int 1
           NOP
INT2:      RETE                    ;Int 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                                ;Int4
      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       *,AR1
      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                ;16 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/288-61/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 G1            32
# define SINX_X        128

extern int TA=24,RA=24,TB=18,RB=18;
extern int AIC_CTR=G0|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);
```