Guide d'intégration SBPA
séquence binaire pseudo aléatoire

Régulations et Asservissements performants et robustes

ADAPTECH

4, rue du Tour de l'Eau
38400 Saint Martin d'Hères, France

tél.: 04 76 51 52 77 ;
Fax: 04 76 42 84 16
e-mail: info@adaptech.com

Tables des matières

Chapitre 1

Intérêt de la SBPA

1- Intérêt de la SBPA

La conception d'un système de régulation performant et robuste nécessite de connaître le modèle dynamique du procédé, qui décrit la relation entre les variations de la commande et les variations de la mesure.

Le modèle dynamique peut être déterminé par identification directe.

- La méthode classique type « réponse en échelon » nécessite des signaux d'excitation de grandes amplitudes, sa précision est réduite, et ne permet pas la validation du modèle.

- Les méthodes actuelles, avec algorithmes d'identification récursifs sur micro ordinateurs offrent une meilleure précision et fonctionnent en boucle ouverte ou fermée avec des signaux d'excitation de très faible amplitude (0,5 à 5 % du point de fonctionnement), et riches en fréquence.

La SBPA (Séquence Binaire Speudo Aléatoire) est un signal formé d'impulsions rectangulaires modulées aléatoirement en longueur, qui approxime un bruit blanc discret, donc riche en fréquence et de valeur moyenne nulle, ne modifiant pas le point de fonctionnement du procédé.

Facile à générer, elle est couramment utilisée dans les procédures d'identification.

Intégrée au même titre que le régulateur RST, dans la cible programmable, la SBPA permet alors une identification en ligne d'un modèle du procédé, et donc un réglage en ligne du régulateur, à partir du poste de supervision, par exemple.

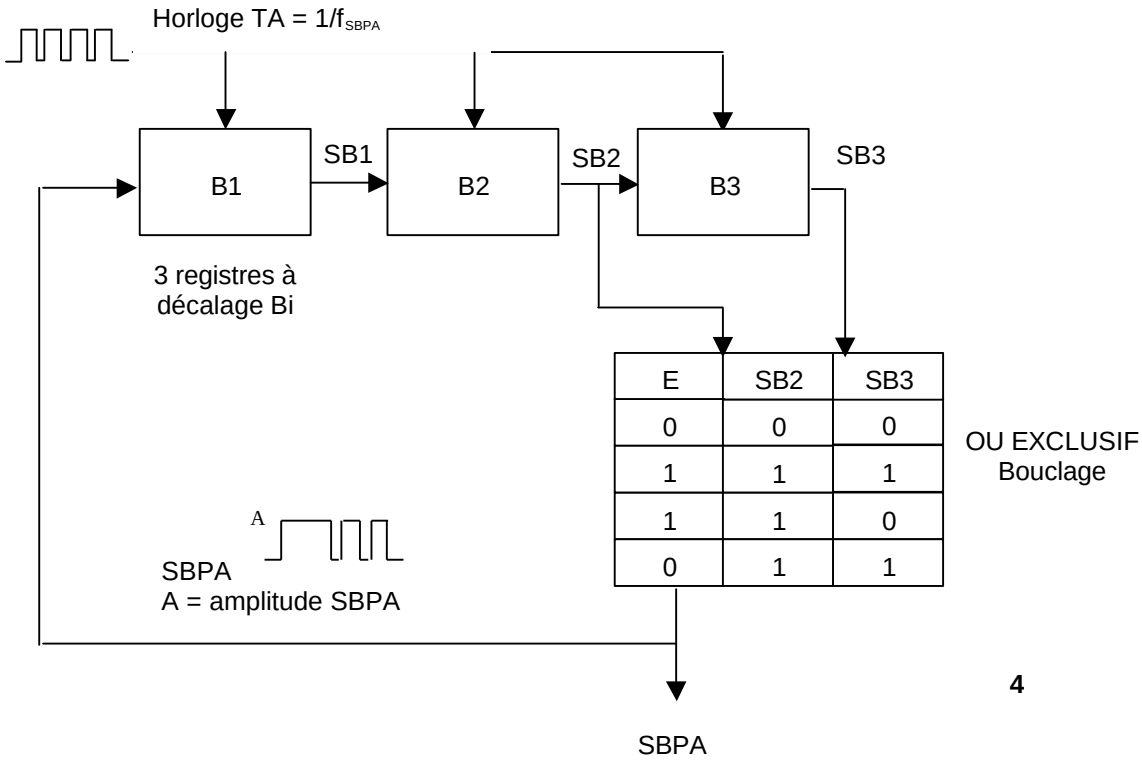
Chapitre 2

Analyse de la SBPA

1- Registre de décalage

La SBPA est un signal pseudo aléatoire car elle est caractérisée par une « longueur de séquence » à l'intérieur de laquelle les variations de la largeur des impulsions varient aléatoirement, mais, sur un grand horizon de temps, elles sont périodiques, la période étant définie par la longueur de la séquence (voir figure 1).

La SBPA est générée à l'aide de registres à décalage (réalisés en matériel ou logiciel) bouclés. La longueur maximale d'une séquence est 2^N-1 où N est le nombre de cellules du registre à décalage. La figure 1 présente la génération d'une SBPA de longueur 7 = 2^3-1 obtenue à l'aide d'un registre à décalage ayant 3 cellules.



Chronogramme :

T_A	SB1	SB2	SB3	E	SBPA = A x E
0	1	1	1	0	- A
$1T_A$	0	1	1	0	- A
$2T_A$	0	0	1	1	+ A
$3T_A$	1	0	0	0	- A
$4T_A$	0	1	0	1	+ A
$5T_A$	1	0	1	1	+ A
$6T_A$	1	1	0	1	+ A
$7T_A=0T_A$	1	1	1	0	
$8T_A=1T_A$	0	1	1	0	
…					
…					

Figure 1

Le tableau 1.1 donne pour différents nombres N de cellules, la structure des bouclages permettant de générer des SBPA de longueur $L= 2^N – 1$.

.

Nombre de cellules N	Longueur de la séquence $L = 2^N - 1$	Bits additionnés Bouclage Bi et Bj
2	3	1 et 2
3	7	2 et 3
4	15	3 et 4
5	31	3 et 5
6	63	5 et 6
7	127	4 et 7
8	255	2, 3, 5 et 8
9	511	5 et 9
10	1023	7 et 10
11	2047	9 et 11

Tableau 1.1 Génération des S.B.P.A. de longueur maximale

Notons aussi un élément caractéristique très important des SBPA : la durée maximale d'une impulsion de la SBPA est égale à N (nombre de cellules).

Cette propriété intervient dans le choix des SBPA pour l'identification.

2- Dimensionnement

Pour bien identifier le gain statique du procédé, il faut que la durée d'au moins une des impulsions (par exemple l'impulsion de durée maximale) soit supérieure au temps de montée t_M du procédé. La durée maximale d'une impulsion étant $N.T_e$, il résulte la condition :

$$N.Te > t_M \qquad (2.1)$$

qui est illustrée dans la figure (2.1) :

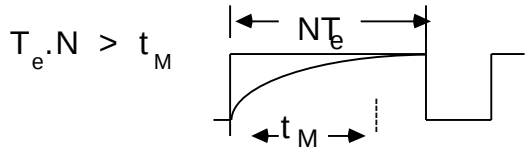


Figure .2.1 Choix de la durée maximale d'une impulsion de la SBPA

A partir de la condition (2.1), on détermine N et donc la longueur de la séquence 2^N-1.

D'autre part, pour balayer tout le spectre de fréquences, il faut que la longueur d'un essai soit au moins égale à la longueur de la séquence. Dans beaucoup de cas, on choisit la durée de l'essai (L) égale à la longueur de la séquence. Si la durée de l'essai est spécifiée, il faut donc s'assurer que :

$$2^N\text{-}1 \,.\, T_e < L \quad (L = \text{durée de l'essai}) \qquad (2.2)$$

A noter que la condition (2.1) peut conduire à des valeurs assez grandes de N correspondant à des longueurs de séquence de durée prohibitive. Soit parce que T_e est très grand, soit parce que le système à identifier risque d'évoluer pendant la durée de l'essai.

C'est la raison pour laquelle dans beaucoup de situations pratiques, on choisit comme fréquence d'horloge pour la SBPA un sous multiple de la fréquence d'échantillonnage.

Si :

$$f_{SBPA} = \frac{f_e}{p} ; p = 1, 2, 3..... \qquad (2.3)$$

alors la condition (2.1) devient

$$p.N.Te > t_M \qquad (2.4)$$

Cette approche est plus intéressante que l'allongement de la longueur de la séquence (augmentation de N). En effet, si on passe de N à N' = N+1 la durée maximale d'une impulsion passe de N Te à (N+1) Te mais la durée de la séquence double L' = 2L. Par contre si on choisit $f_{SBPA} = f_e/2$, la durée maximale d'une impulsion passe de N T_e à 2 N T_e pour une durée de la séquence doublée L' = 2L.

De la comparaison des deux approches, il résulte que la deuxième approche (division de fréquence) permet d'obtenir une impulsion de durée plus grande pour une durée identique de la séquence et donc de l'essai. Si on note par p l'entier diviseur de fréquence, on a dans le cas de la division de la fréquence d'horloge (d_{max} = durée de l'impulsion maximale).

$d_{max} = p\ N.T_e$; $L' = p\ L$; p = 1, 2, 3...

En augmentant N par (p-1) et donc la longueur de la séquence, on a :

$d_{max} = (N+p-1)\ T_e$; $L' = 2^{(p-1)}\ L$; p = 1, 2, 3...

3- Amplitude

L'amplitude de la SBPA peut être très faible, mais elle doit être supérieure au niveau du bruit résiduel. Si le rapport signal/bruit est trop faible, il faut allonger la longueur de l'essai pour pouvoir obtenir une bonne estimation des paramètres.

Notons que, dans de nombreuses applications l'augmentation significative du niveau de la SBPA n'est pas souhaitable à cause du caractère non linéaire des procédés à identifier (nous nous intéresserons à l'identification d'un modèle linéaire autour d'un point de fonctionnement).

Une valeur typique de l'amplitude est de 0,5 % à 5 % de la valeur du point de fonctionnement sur lequel est appliquée la SBPA.

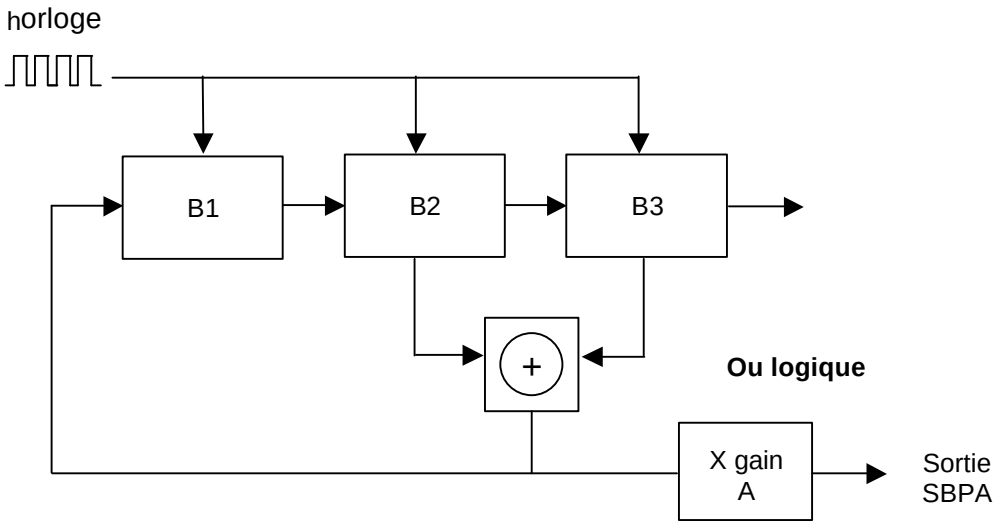
Chapitre 3

Algorithme SBPA

1- Présentation sous forme de blocs fonctionnels

La présentation découle directement de la structure présente au paragraphe 2-1 ci-avant avec les registres à décalages.

Exemple pour N = 3 registres : B1, B2 et B3



2- Présentation sous forme d'un programme

SBPA appliquée autour d'un point de fonctionnement.

Description des variables:

N = nombre de registres,
REG(.) =registre à décalage,
K1, K2 =bits à additionner du registre,
NE = nombre d'échantillons à acquérir,
P = entier diviseur de fréquence,
UI = point de fonctionnement du processus,
U(.) = vecteur contenant les entrées (excitations) à envoyer aux processus

AS = amplitude de la SBPA

- Programme :

```
'initialisation des bits à additionner
K1 = N - 1
K2 = N
 IF N=5 THEN  K1 = 3
IF N=7 THEN  K1 = 4
IF N=9 THEN  K1 = 5
IF N=10 THEN  K1 = 7
IF N=11 THEN  K1 = 9

    ' initialisation du registre à décalage
    FOR I=1 TO N
            REG(I) = 1
    NEXT I
    ' NE = multiple de P
    NE = INT(NE/P) * P

    ' initialisation du vecteur U par groupes de P éléments
    FOR K=1 TO NE  STEP  P
            ' UIU = 1 or -1 (bit du registre à décalage)
            UIU = -REG(K1) * REG(K2)
            IF N=8 THEN  UIU = -REG(2) * REG(3) * REG(5) * REG(8)
```

```
' initialisation de P éléments du vecteur U
FOR J=K TO (K+P-1)
U(J) = UIU * AS + UI     ' (+/-1 * amplitude SBPA) + point de fonctionnement
NEXT J

        ' décalage du registre
        FOR I=N TO 2  STEP -1
                REG(I) = REG(I-1)
        NEXT I
        REG(1) = UIU

NEXT K
```

Chapitre 4

Intégration de la SBPA dans le programme principal RST

L'algorithme de la SBPA décrit au chapitre 3 peut être écrit dans tout langage informatique (C, C++, Basic, Pascal, Fortran) en utilisant des opérations simples d'addition, multiplication et décalage de registres.

Le programme ci-après donne un exemple complet d'intégration, incluant un programme principal pour l'Automate LT 160 LAI, l'algorithme de régulation RST et l'algorithme SBPA (voir aussi le Guide d'Intégration RST pour la partie régulateur).

Il permet de réaliser les mesures avec SBPA nécessaires à l'identification, en boucle ouverte (ou en boucle fermée, en ligne, avec ou sans le régulateur RST intégré, et avec SBPA sur la consigne ou sur la commande).

Il est écrit en C++, et peut aisément être transcrit en un autre langage, et / ou pour une autre cible.

La partie correspondant à la SBPA est représentée en caractères gras.

1- Déclaration des constantes, variables et fonctions utilisées

1-1 Déclaration des constantes

- **BO** = 1 : excitation par sbpa, du procédé en Boucle Ouverte (sans régulation).
- **BF** = 2 : excitation par sbpa, du procédé en Boucle Fermée (avec régulation).
- **NO** = 0 : pas de signal sbpa additionné.
- **ACTION** = 1 : signal sbpa additionné sur la commande (en BO ou en BF).
- **SETPOINT** = 2 : signal sbpa additionné sur la consigne (en BF).
- **NVALSBPA** = 254 : nombre de valeur contenu dans le signal sbpa égal à p * (2^N -1).

1-2 Déclaration des variables globales

- **Flag_boucle = …** de type entier *: état de la boucle de régulation (BO ou BF).*
- **Flag_sbpa = …** de type entier *: indicateur d'application de la sbpa (NO, ACTION et SETPOINT).*

- **Uinit = …** de type réel *: point de fonctionnement du processus.*
- **Ampli = …** de type réel *: amplitude du signal.*
- **Decal = …** de type réel *: offset du signal.*
- **LgReg = …** de type entier *: longueur du registre N.*
- **Divi = …** de type entier *: diviseur de fréquence p.*

- **attente = …**de type entier *: tempo d'application de la sbpa (en période d'échantillonnage).*
- **compteur** de type entier *:compteur d'itération (en nombre de période d'échantillonnage).*
- **Signal_sbpa** tableau de réel de dimension NVALSBPA *: signal de sbpa.*

1-3 Déclaration des procédures et fonctions

Procédure **creer_sbpa**(ValUnit, ValAmpli, ValDecal, ValLgReg, ValDivi, nombre_valeur, signal_sbpa)

Paramètres d'entrée :

- **ValUinit** de type réel : *point de fonctionnement du processus.*
- **ValAmpli** de type réel *: amplitude du signal.*
- **ValDecal** de type réel *: offset du signal.*
- **ValLgReg** de type entier *: longueur du registre N.*
- **ValDivi** de type entier *: diviseur de fréquence p.*
- **nombre_valeur** de type entier *: dimension du tableau de valeur de la sbpa.*

Paramètres de sortie :

- **signal_sbpa** tableau de réel de dimension nombre_valeur *: signal de sbpa..*

Rôle : génère la Séquence Binaire Pseudo-Aléatoire dans un tableau.

2- Structure du programme principal intégrant la SBPA

Exemple d'intégration de la SBPA (caractères gras) dans le programme principal RST.

```
Si    (premier appel du régulateur)   alors        : commutation Manu / Auto
|     status = INITCON                             : passage en phase d'initialisation
|
|     Si    (Flag_sbpa ¹ NO) alors                  : création tableau sbpa
|     |     Creer_sbpa(Uinit, Ampli, Decal, LgReg, Divi, NVALSBPA, Signal_sbpa)
|     Finsinon
|
|     compteur = 0                                  : initialisation compteur
Finsi
Pour  (chaque période d'échantillonnage)   faire    : Si temps = N.Te (N entier)
|     compteur = compteur + 1                       : actualisation compteur|
|     Si    (Flag_boucle = BF)   alors              : en boucle fermée
|     |     Si    status = INITCON   alors          : phase d'initialisation
|     |     |     acquisition_mesure(Measure)       : lecture de la mesure
|     |     |     acquisition_commande(Actionmanu)  : lecture de la commande appliquée
|     |     |     init_stockage(Measure, Actionmanu, Measure)   : initialisation du RST
|     |     |     status = RUNCON                   : passage en phase de calcul
```

| Sinon

| | precalcul_rst() *: 1er tiers de calcul du RST*

| | acquisition_mesure(Measure) *: lecture de la mesure*

| | acquisition_consigne(Setpoint) *: lecture de la consigne*

| | **Si ((Flag_sbpa = SETPOINT) Et (0 £ compteur-attente £ NVALSBPA) alors**
| | **Setpoint = Setpoint + signal_sbpa[compteur-attente]** : ajout **sbpa sur la consigne**

| | **Finsi**|

| | Action = execalcul_rst(Setpoint, Measure) : *2ème tiers de calcul du RST*

| | **Si ((Flag_sbpa = ACTION) Et (0 £ compteur-attente £ NVALSBPA)) alors**

| | | **Action = Action + signal_sbpa[compteur-attente]** *: ajout **sbpa sur la commande***

| | **Finsi**

| | application_commande(Action) *: envoi de la commande en boucle fermée*

| | postcalcul_rst(Action) *: 3ème tiers de calcul du RST*

| Finsinon

Sinon *: **en boucle ouverte***

| **Si ((Flag_sbpa = ACTION) Et (0 £ compteur-attente £ NVALSBPA)) alors**

| | **Action = Action + signal_sbpa[compteur-attente]** *: ajout sbpa*

| Sinon

| | Action = Uinit *: point de fonctionnement*

| Finsinon

| | application_commande(Action) *: envoi de la commande en boucle ouverte*

Finsinon

Finpour

3- Sources C++ du programme principal RST + SBPA

Le programme source C++ présenté ici est un exemple complet et opérationnel de programme principal qui appelle les différentes fonctions et procédures RST décrites ci-avant.

Il a été implémenté sur une cible type automate LT 160 de Leroy Automatique Industrielle, automate programmable en C++ ou avec le progiciel ISaGRAF. Le "procédé" est un bras flexible articulé sur ressorts, en rotation sur une plate-forme pivotante ; il est équipé d'un moteur CC et de deux capteurs de position angulaire.

Représentation :

Le code en caractères :

- **gras** correspond à la fonction SBP
- *italiques* correspond à la fonction RST
- normaux correspond à l'environnement spécifique à la cible (LT160 LAI dans le cas présent)

```
/*********************************************************************
      */
/*  Projet :       Intégration d'un algorithme de régulation RST      */
/*                 avec la possibilité de superposer un signal SBPA   */
/* (Séquence Binaire Pseudo-Aléatoire) pour l'identification          */
/*                                                                    */
/* Auteur :       ADAPTECH         Programme RST_SBPA.c               */
/* Date :         Octobre 2000                                        */
/*                                                                    */
/* Description : Programme principal pour l'intégration d'un          */
/*               régulateur RST                                       */
/*              sur un Automate LT160 Leroy Automatique Industrielle  */
/*               pour commande d'un Bras flexible articulé            */
/*                                                                    */
/*********************************************************************/

//--- Fichiers d'en-tête ------------------------------------------

#include <stdlib.h>
#include<conio.h>
#include<stdio.h>


//--- Automate Lt160 Leroy Automatique
#include <alarm.h>
#include <timebase.h>
#include <cpuled.h>
//Carte 8E/4S Ana AIO320
#include <ioboardm.h>
#include <AIO320.h>
//Timer
#include <waveform.h>
// fonctions de calcul
#include <math.h>


//--- Déclaration des constantes ----------------------------------

#define POINT_VOLT 0.0003051850948  // Facteur de conversion:32767 points
pour 10V

#define DEGMAX      10      // Nombre de coefficient max du RST
#define INITCON     1         // Etat du régulateur: Initialisation
#define RUNCON      2       // Etat du régulateur: Calcul

#define BO          1       // excitation par sbpa du procédé en Boucle
Ouverte (sans régulation)
#define BF          2        // excitation par sbpa du procédé en Boucle
Fermée (avec régulation)
#define NO          0       // pas d'application de sbpa
#define ACTION      1           // signal sbpa appliquée sur la commande
(en BO ou en BF)
```

```
#define SETPOINT    2         // signal sbpa appliquée sur la consigne (en
BF)
#define NVALSBPA    508     // nombre de valeur contenue dans le signal
sbpa égal à p * (2^N-1)

//--- Déclaration des booléens

# define FALSE 0
# define TRUE 1

//--- Déclaration des variables concernant la sbpa ----------------------
// Indicateur de boucle :
//                       - BO (boucle ouverte)
//                       - BF (boucle fermée)
int flag_boucle = BF;

// Indicateur d'application de sbpa :
//                                     - NO (pas d'application)

//                                     - ACTION (sur la commande)
//                                     - SETPOINT (sur la consigne)
int flag_sbpa = NO;

// Valeur initiale du signal (point de fonctionnement)
float Uinit = 0;

// Amplitude du signal
float Ampli = 1.0;

// Offset du signal
float Decal = 0;

// longueur du registre N
int LgReg = 7;

// diviseur de fréquence p
int Divi = 2;

// Temps d'attente en période d'échantillonnage avant application du
signal sbpa
int attente = 50;

// Compteur d'itération (en nombre de période d'échantillonnage)
int compteur;

// tableau contenant le signal sbpa de dimension NVALSBPA =  p * (2^N-1)
float signal_sbpa[NVALSBPA];

//--- Déclaration des variables globales RST -----------------------------
-

// Etat du régulateur
int status;
```

```
// Mesure et consigne
float Measure ;
float Setpoint;
// Commande et Commande Manu
float Action, Actionmanu;

//--- Variables RST

// Min, Max et précision du CNA pour la commande
float   umin=-10 ;   //-10V
float   umax= 10 ;   //+10V
float   upre= 0.1; //(0.2/100*2048)*POINT_VOLT ;  résolution de la carte
sur 11 bits
// Sortie de saturation selon profil du 1er ordre de constante de temps
Tsat

// Coefficient . calculer: Ssat= exp(-Te/Tsat)
float Ssat=0.0;  // pas de dynamique: sortie avec un simple gain

//=== Paramètres du régulateur RST :  à saisir ==========================

int Bmdeg=1;
float Bm[2]=    {0.300925,   0.151055 };
int Amdeg=2;
float Am[3]=    {1. ,   -0.676498,   0.128478 };
int Rdeg=4;
float R[DEGMAX]={1.665709, -0.747280, -0.981312,  0.892897, -0.538781 };
int Sdeg=4;
float S[DEGMAX]={1.000000,  -0.490052,  -0.180290, -0.191399, -0.138259 };
int Tdeg=4;
float T[DEGMAX]={10.716965, -23.676476,  18.949705,  -6.769146,   1.070185
};

  int Te=60;    // Période d'échantillonnage Te: en ms

//=======================================================

//--- Variables utilisées pour le calcul du RST -------------------------

// Vecteurs de signaux
float yt[DEGMAX], ut[DEGMAX], yreft[DEGMAX];
float yrt[DEGMAX], urt[DEGMAX];
/* L' indice i correspond au temps t-i, t _tant l'instant actuel
par exemple, ut[1] représente u(t-1), yt[3] représente y(t-3)0 */

float compo_continu;      // composante continue du régulateur
float prepa_cmde;       // valeur intermédiaire pour le calcul
float gain = 1;         // Gain


//--- Déclaration de la carte 8E/4S Ana AIO320
IoBoardM_TS_AIO320 Control_V_Carte1;
```

```
//--- Capteurs de position angulaire sur machine
float Plateforme; // entrée +/-10V EA0  rotation Plateforme
float Bras;       // entrée +/-10V EA1  rotation Bras flexible

//--- Déclaration du timer LT160
WaveForm_TS_Data Control_V_Timer;



//--- Déclaration des procédures et fonctions RST et SBPA

float arrondi(float numb, float precis);
void Creer_sbpa(float ValUinit, float ValAmpli, float ValDecal, int
ValLgReg, int ValDivi,
                                                    int nombre_valeur,float
*signal_sbpa );
void precalcul_rst(void);
float calc_ref(float nvcons);
float calc_cmde(float yref, float mesure);
float execalcul_rst(float consigne, float mesure);
void decal_trajref(float consigne, float yref);
void postcalcul_rst(float commande);
void init_stockage ( float yinit, float uinit, float cinit);


//-------------------------------------------------------------------------
--
//       Programme principal à implémenter dans le système temps réel
//-------------------------------------------------------------------------
--

void main()
{
Alarm_F_Write(false);   // chien de garde Automate LT160

      if (flag_sbpa != NO) {
                  Creer_sbpa(Uinit, Ampli, Decal, LgReg, Divi, NVALSBPA,
signal_sbpa );
      }

 //Insertion de la carte analogique AIO320
      if (IoBoardM_F_InsertLt(&Control_V_Carte1, IoBoardM_V_AIO320, 1) ==
0)
             CpuLed_F_Write (2,1);   // led I/O allumée fixe


  // Déclaration du signal d'échantillonnage et affectation du compteur
      WaveForm_F_Timer(&Control_V_Timer, &TimeBase_V_Millisecond);

 // Période des impulsions d'échantillonnage: Te ms
      WaveForm_F_Preset(&Control_V_Timer, Te);

 // Remise à 0 et lancement du timer
```

```
      WaveForm_F_Restart(&Control_V_Timer);

      status=INITCON; // 1er passage dans le RST
      Actionmanu=0;
      compteur = 0;   // initialisation du compteur d'itération



while(1)     // début du while de la boucle infinie ----------------------
-

{
IoBoardM_F_RefreshInputLt();
if (WaveForm_F_BOutput(&Control_V_Timer)     // Toute les Te ms: calcul de
la commande
                  {
                        compteur ++;                    // actualisation
compteur
            Plateforme =  Control_V_Carte1.Input[0]*POINT_VOLT  ;
                  Bras =        Control_V_Carte1.Input[1]*POINT_VOLT  ;
Measure=Plateforme+(0.418*Bras); // acquisition de la mesure de rotation
du bras

if (flag_boucle == BF)                // En Boucle Fermée
                        {
                        if ((status == INITCON))  // phase
d'initialisation
                              {
init_stockage(Measure, Actionmanu, Measure);   // initialisation du
régulateur RST
                              status = RUNCON;
                        }
                              else                      // phase de
calcul
                              {
precalcul_rst();                    // precalcul du RST (étape 1/3)

                                    if ((flag_sbpa ==
SETPOINT)&&(compteur >=attente)
&&(compteur <= attente+NVALSBPA))
                                                {
Setpoint = Control_V_Carte1.Input[2]*POINT_VOLT; // lit consigne
Setpoint = Setpoint + signal_sbpa[compteur-attente]; // ajout sbpa
                                                }
                                                else
                                                {
Setpoint = Control_V_Carte1.Input[2]*POINT_VOLT; // lit consigne
                                                }

Action = execalcul_rst(Setpoint, Measure);   // calcul du RST (étape 2/3)
```

```
                                        if ((flag_sbpa == ACTION)&&(compteur
>= attente)
      &&(compteur <= attente+NVALSBPA))
                                        {
Action = Action + signal_sbpa[compteur-attente];    // ajout sbpa
Control_V_Carte1.Output[0]= Action / POINT_VOLT ;   // action en BF

}
                                        else
                                        {
Control_V_Carte1.Output[0]= Action/POINT_VOLT ;       //  action BF
                                        }

                              postcalcul_rst(Action);   // postcalcul du
RST (étape 3/3)

                              }           // fin du if de régulation
                  }
                        else                                    // en
Boucle Ouverte
                        {
                        if ((flag_sbpa == ACTION)&&(compteur >= attente)
      &&(compteur <= (attente+NVALSBPA)))
                              {
                                        Action =
signal_sbpa[compteur-attente];  // ajout sbpa
                              Control_V_Carte1.Output[0]= Action /
POINT_VOLT ;                            // action BO

      }
                              else
                              {
                                        Action = Uinit;           //
point de fonctionnement
                                        Control_V_Carte1.Output[0]=
Action / POINT_VOLT ;                   // action BO
                              }
                        }
            }                         // fin du if d'impulsion
      IoBoardM_F_RefreshOutputLt();
      };                              // fin du while de la boucle infinie
----------------


}  //      === fin du Programme principal ===


//*************************************************************************
**
//   FONCTIONS et PROCEDURES pour l'ALGORITHME du REGULATEUR RST et SBPA
//*************************************************************************
**
```

```
//-------------------------------------------------------------------------
-
// Fonction : arrondi
// appelée par la fonction "calc_cmde"
// (pour prendre en compte la précision du CNA pour la commande)
//-------------------------------------------------------------------------
--
float arrondi(float numb, float precis)
{
    float invpre;

    if (precis == 0.0)
      return (numb);
    invpre = 1.0/precis;
    if ((numb-floor(invpre*numb)/invpre)>((ceil(invpre*numb)/invpre)-
numb))
      return (ceil(invpre*numb)/invpre);
    else
      return (floor(invpre*numb)/invpre);
}
//-------------------------------------------------------------------------
-
// Fonction : Creer_sbpa( )
// Génère la Séquence Binaire Pseudo-Aléatoire dans un tableau.
//-------------------------------------------------------------------------
--
void Creer_sbpa(float ValUinit, float ValAmpli, float ValDecal, int
ValLgReg,
                int ValDivi,  int nombre_valeur, float *signal_sbpa )
{

/* Les paramètres d'entrée sont :
 ValUinit : Valeur initiale du signal
 ValAmpli : Amplitude du signal
 ValDecal : Offset
 ValLgReg : valeur du registre N (entre 2 et 11)
 ValDivi : valeur du diviseur de fréquence p (entre 1 et 3)
 nombre_valeur : nombre de valeur de sbpa = p*2^(N-1)

Après création du signal, le paramètre de sortie est  :
    le vecteur signal_sbpa qui contient les valeurs de la sbpa à appliquer
    à chaque instant d'échantillonnage.
*/
    int     k1, k2, i, j, uiu, sbpa[11];

    k1 = ValLgReg-1;
    k2 = ValLgReg;
    if (ValLgReg == 5)  {k1 = 3;}
    if (ValLgReg == 7)  {k1 = 4;}
    if (ValLgReg == 9)  {k1 = 5;}
    if (ValLgReg == 10) {k1 = 7;}
    if (ValLgReg == 11) {k1 = 9;}

    for(i=1; i<=11; i++)
```

```
        sbpa[i] = 1;

    i = 1;
    while(i <= nombre_valeur){
        uiu = -sbpa[k1]*sbpa[k2];
        if (ValLgReg == 8)
            uiu = -sbpa[2]*sbpa[3]*sbpa[5]*sbpa[8];
        j = 1;
        while(j<=ValDivi){
            signal_sbpa[i-1] = uiu * ValAmpli + ValDecal + ValUinit;
            i++;
            j++;
        }
        for(j=ValLgReg; j >= 2; j--)
            sbpa[j] = sbpa[j-1];
        sbpa[1] = uiu;
    }
}


//=== Fonction appel ===

//-------------------------------------------------------------------------
--
// Fonction : init_stockage
// appelée une fois par la fonction "control"
// (durant la phase d'initialisation)
//-------------------------------------------------------------------------
--

void init_stockage ( float yinit, float uinit, float cinit)
{
   int j;
   float sigmaR, sigmaT, sigmaS;

   sigmaR = 0.;
   for (j = 0; j <= Rdeg; j++){
      yt[j] = yinit;
      sigmaR += R[j];
   }
   sigmaS = 0.;
   for (j = 0; j <= Sdeg; j++){
      ut[j] = uinit;
      sigmaS += S[j];
   }
   sigmaT = 0.;
   for (j = 0; j <= Tdeg; j++){
      yreft[j] = cinit;
      sigmaT += T[j];
   }
   compo_continu = gain * (sigmaR-sigmaT) * yinit + sigmaS * uinit;
```

```
   for (j = 0; j <= Bmdeg; j++)
   {  urt[j] = cinit; }

   for (j = 0; j <= Amdeg; j++)
   {  yrt[j] = cinit; }
   S[1] = S[1] - Ssat;
}


//-------------------------------------------------------------------------
-
// Fonction : precalcul_rst
// appelée à chaque période d'échantillonnage par la fonction "control"
// (avant l'acquisition des mesures)
//-------------------------------------------------------------------------
-

void precalcul_rst(void)
{
   float somme;
   int  j;

   somme = compo_continu;
   for (j=1; j<=Tdeg; j++)
   {  somme += gain * T[j]*yreft[j]; }
   for (j=1; j<=Rdeg; j++)
   {  somme -= gain * R[j]*yt[j]; }
      for (j=1; j<=Sdeg; j++)
   {  somme -= S[j]*ut[j]; }
   prepa_cmde = somme;
   return;
}


//-------------------------------------------------------------------------
--
// Fonction : calc_ref
// utilisée par la fonction "execalcul_rst"
//-------------------------------------------------------------------------
--

float calc_ref(float nvcons)
{
   int j;
   float somme;

   somme = Bm[0] * nvcons;
   for (j = 1; j <=Bmdeg; j++)
       { somme += Bm[j] * urt[j]; }
   for (j = 1; j <=Amdeg; j++)
       { somme -= Am[j] * yrt[j]; }
```

```
   return(somme);
}
```

```
//---------------------------------------------------------------------
// Fonction : calc_cmde
// utilisée par la fonction "execalcul_rst"
//---------------------------------------------------------------------
--

float calc_cmde(float yref, float mesure)
{
   float somme;

   somme = prepa_cmde;
   somme -= gain * R[0]*mesure;
   somme += gain * T[0]*yref;
   somme = somme / ( S[0] + Ssat * ut[1] ) ;

   somme = arrondi(somme, upre);

   if (somme < umin) { return umin; }

   if (somme > umax) { return umax; }

   return somme;
}


//---------------------------------------------------------------------
// Fonction : execalcul_rst
// appelée à chaque période d'échantillonnage par la fonction "control"
// (après l'acquisition des mesures et avant l'envoi de la commande)
//---------------------------------------------------------------------

float execalcul_rst(float consigne, float mesure)
{
   float commande;

   yt[0] = mesure;
   urt[0] = consigne;
   yreft[0] = calc_ref(consigne);
   commande = calc_cmde(yreft[0], mesure);
   return commande;
}
```

```
//---------------------------------------------------------------------
// Fonction : decal_trajref
// utilisée par la fonction  "postcalcul_rst"
//---------------------------------------------------------------------

void decal_trajref(float consigne, float yref)
{
   int j;

   for (j=Bmdeg; j>1; j--)
      { urt[j] = urt[j-1]; }
   for (j=Amdeg; j>1; j--)
      { yrt[j] = yrt[j-1]; }

   urt[1] = consigne;
   yrt[1] = yref;
}

//---------------------------------------------------------------------
// Function : decal_obser
// utilisée par la fonction  "postcalcul_rst"
//---------------------------------------------------------------------

void decal_obser(float yref, float mesure, float commande)
{
   int j;

   for (j=Rdeg; j>1; j--)
{ yt[j] = yt[j-1]; }
   for (j=Sdeg; j>1; j--)
      { ut[j] = ut[j-1]; }
   for (j=Tdeg; j>1; j--)
      { yreft[j] = yreft[j-1]; }

   yt [1] = mesure;
   ut [1] = commande;
   yreft [1] = yref;
}
```

```
//-----------------------------------------------------------------------
--
// Fonction : postcalcul_rst
// appelée à chaque période d'échantillonnage par la fonction "control"
// (après l'application de la commande)
//-----------------------------------------------------------------------
----------------------

void postcalcul_rst(float commande)
{
   decal_trajref(urt[0], yreft[0]);
   decal_obser(yreft[0], yt[0], commande);
   return;
}

//***********************************************************************
**
//                              FIN DE L'ALGORITHME RST + SBPA
//***********************************************************************
**
```

La disquette contenant le fichier source RST_SBPA.c est disponible sur simple demande

ADAPTECH
4 rue du Tour de l'Eau
38400 Saint Martin d'Hères

Tél. : 04 76 51 52 77
Fax : 04 76 42 84 16
E-mail : info@adaptech.com