ASSOCIATION POUR LE DEVELOPPEMENT DE L'ENSEIGNEMENT
ET DE LA RECHERCHE EN SYSTÉMATIQUE APPLIQUEE

MELISSA

Memorandum of Understanding
ECT/FG/MMM/97.012

CONTRACT ESA-ESTEC/ADERSA
Purchase Order n° 171 686 of 16/07/1997

TECHNICAL NOTE : 38.2

Final validation of the production control of the 7 litres photoautotrophic compartment

Version : 1
Issue : 0

LECLERCQ J.-J.

May 1998

10, rue de la Croix Martre
91873 PALAISEAU Cedex
Téléphone : (33) 01 60 13 53 53
Télécopie : (33) 01 69 20 05 63
Courriel : adersa@adersa.asso.fr

aadersa

Document Change Log

Version	Issue	Date	Observation
0	0	May 1998	Draft
1	0	May 1998	Original version

TABLE OF CONTENTS

1. Introduction

1.1. Control strategy

A first validation of the control strategy of the Spirulina compartment was done at ESTEC in 1995. It concerns only the control of the biomass production by acting on the light intensity.

The main principles of this strategy is reminded hereafter (diagram of figure 1) :

Level 0 : It consists in the regulation of the light intensity. The measure is the light intensity in the center of the reactor E_b. Level 0 calculates the action to apply to the potentiometer of the lamps to regulate the light intensity in the center of the reactor. This action is calculated with a classical PI controller.

Level 1 : It consists in the regulation of biomass production by action on the light intensity. The control law is a non linear predictive control law, which uses the non linear knowledge model and which consists in applying on it some scenarios of radiant flux values F_r during the prediction horizon (see TN 24.1).

The available model allows to calculate a radiant flux F_r, which is converted in to a corresponding value E_b setpoint of light intensity in the center of the reactor, through a non-linear model, using the measure of the biomass concentration.

Level 2 : The role of this level is the optimisation of setpoints, with respect to constraints. It calculates feasible production and flow setpoints in order to respect the constraints and to optimize the functioning.

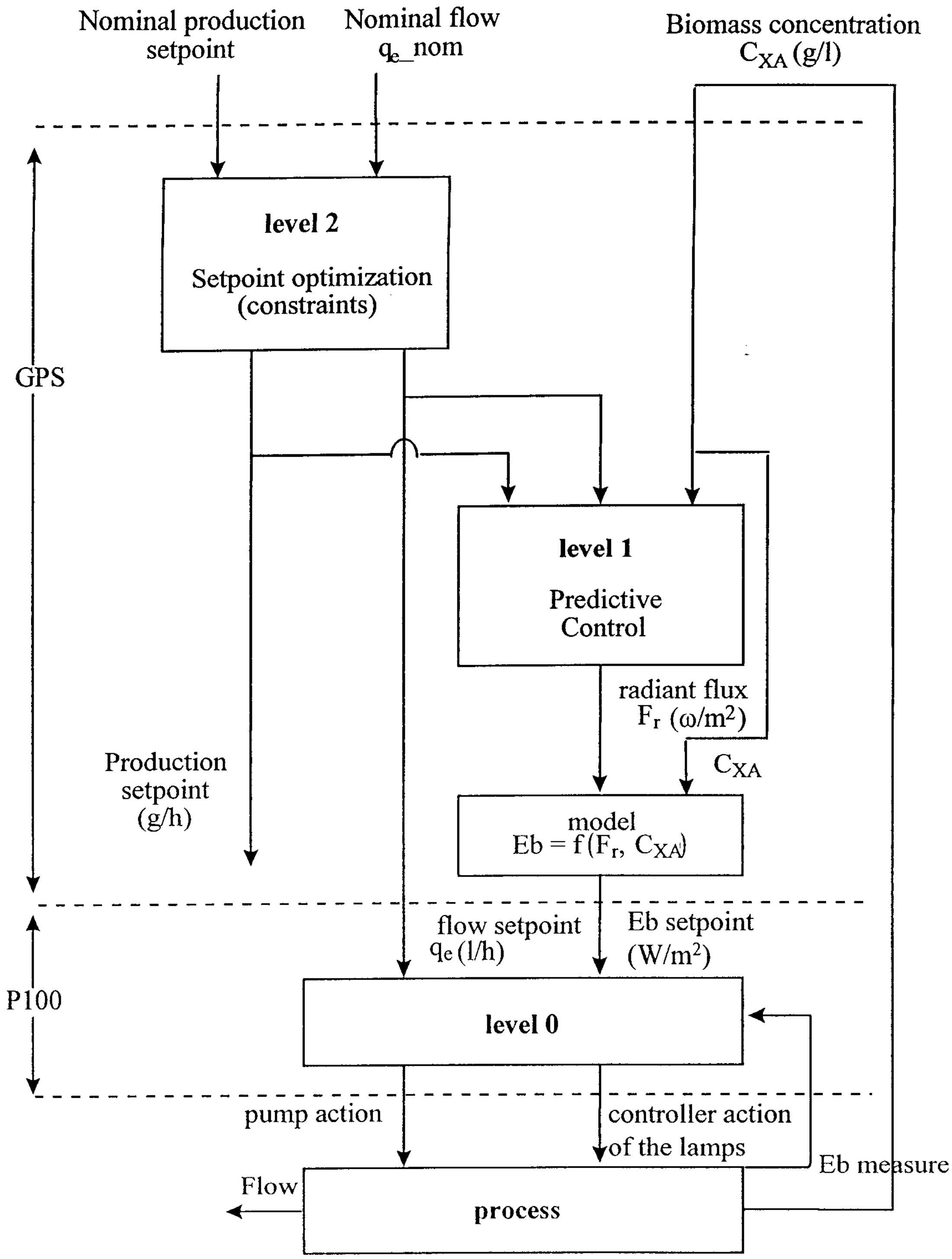


Figure 1 - Hierarchical structure of the control law from TN 24.2 ADERSA 1995

1.2. Experimental results

The previous control strategy was tested at ESTEC. The figure 2 shows an example of the tests. A stactic bias of production control can be seen ; it is about 3 % of the setpoint.

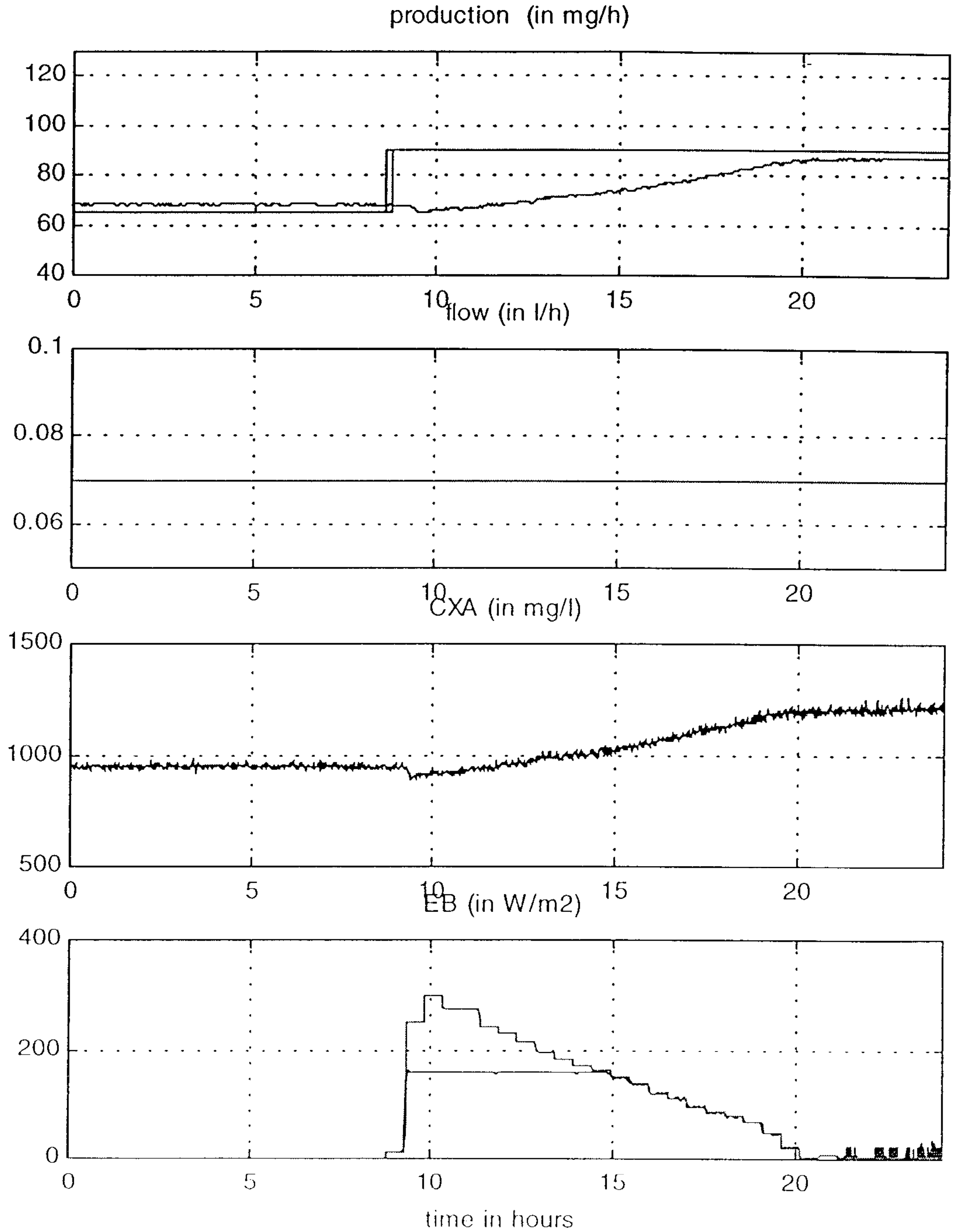


Figure 2 : Experimental results on April 13th 1995 at ESTEC

2. Improvement and final validation

After this experiment at ESTEC in 1995, the Spirulina reactor was installed in Barcelona, at UAB.

Then, it was decided :

1. to eliminate the level 0 (the light power controller) and to devote to level 1 the role of computing directly the action of the potentiometer of the lamps ;
2. to cancel the static bias.

2.1. Elimination of the light power controller

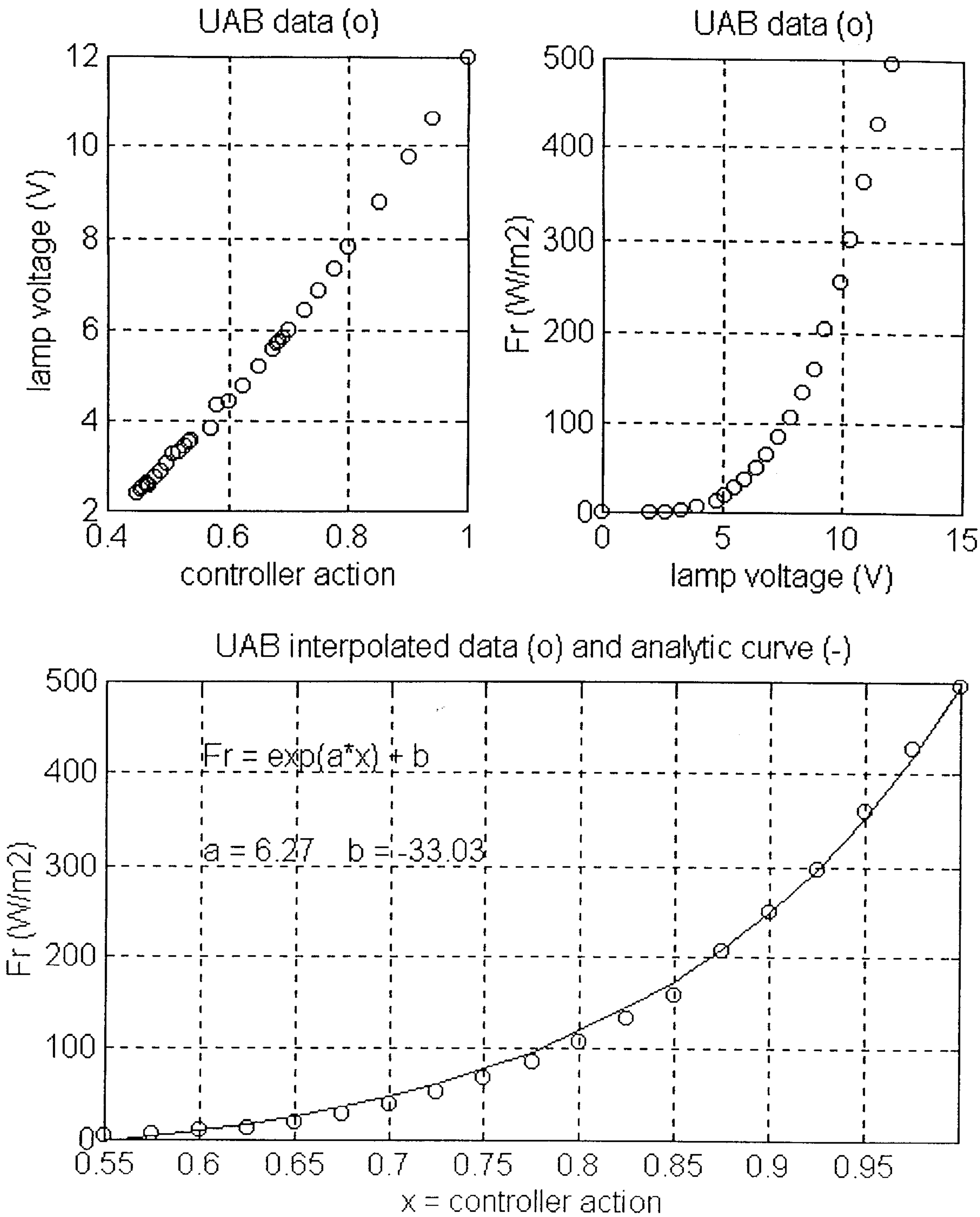
From UAB data (figure 3), it was built an analytic relation between the controller action, x, and the light power, F_r :

$$F_r = \exp(a \cdot x) + b$$

with $a = 6.270$ $b = -33.03$

F_r in W/m^2

x : scalar value between 0 and 1



UAB data on 16th September 1997

Figure 3 : Relation between Controller_action and Fr

The modified control law was tested at UAB in September 1997 (figure 4). As it was the case at ESTEC, a static bias appeared (about 3 % of the setpoint).

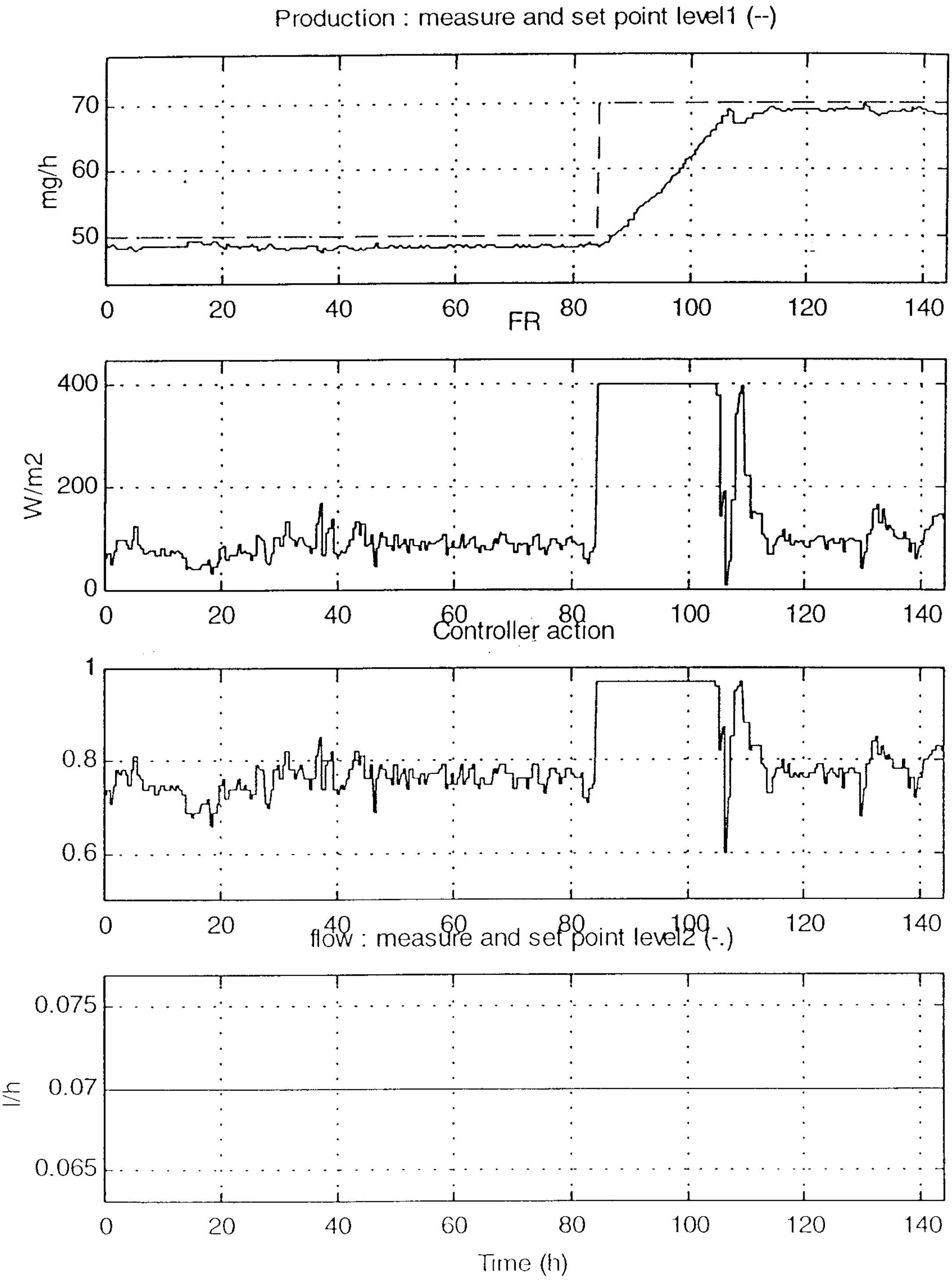


Figure 4 : UAB data from Sept 26th 0h00 to Oct 1st midnight

2.2. Elimination of the static bias

Principle :

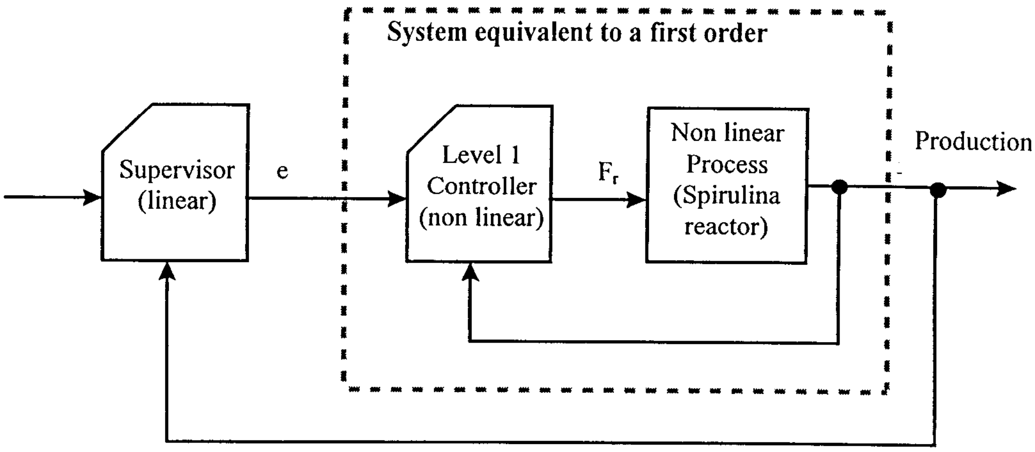


Figure 5

The internal model of the linear supervisor is a first order transfer with a gain equal to 1 and a time response equal to the closed loop time response of the level 1 controller. The theory of control shows that the closed loop system of figure 5 has no static bias.

As Fr is constrained between $F_{r_}$min and $F_{r_}$max, a set of the corresponding constraints has to be computed for the output of the supervisor, e.

The expressions of e and its constraints are detailled in Annex 1.

Robustness :

Different kinds of mismatch between the process and the model of the controller had been studied in simulation. Figure 6 shows the case where $F_{r_}$process = $F_{r_}$model - 50 (in W/m^2) : the bias is completely cancelled and a slight overshoot appears.

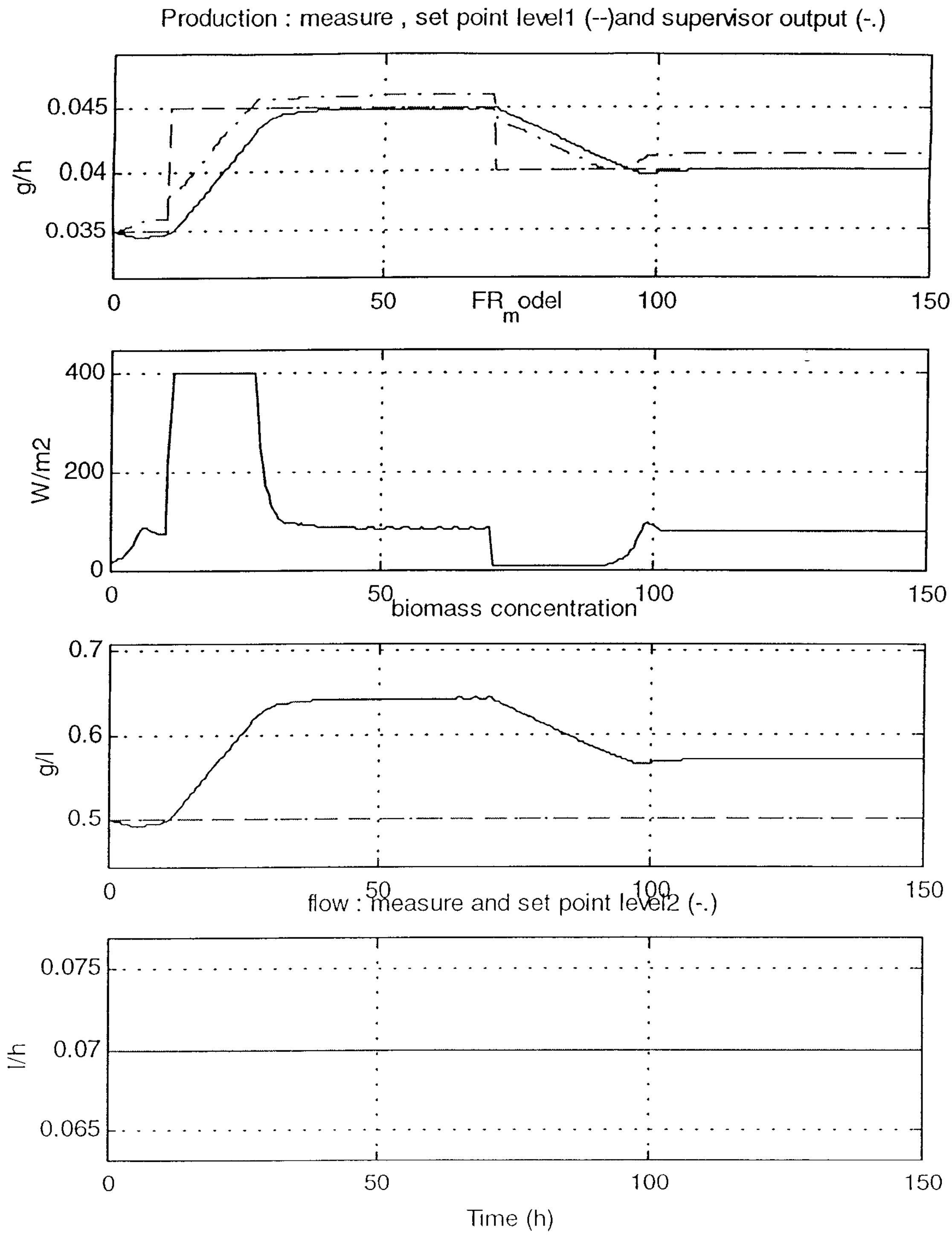


Figure 6 : Simul. of mismatched controller (Fr_process = Fr_model - 50)

Experimental result :

This supervisor has been tested at UAB in October 1997 (figure 7).

At the moment when the supervisor was loaded in the Global Purpose Station (at time about 18 h on figure 7), the controller action increases immediately so that the production reachs its set point in about 12 hours, with no more bias.

At time t = 35 h, there is a sudden variation of the production due to bad working of the sensor of biomass concentration. That results in a sudden variation of the controller action from the minimum 0.6 to the maximum 0.97 corresponding to the extreme values of F_R, 10 and 400 W/m^2.

Between times t = 65 h and t = 70 h, there is a misfunctionning of the GPS. Then the control of the production is quite good, with no bias and no overshoot.

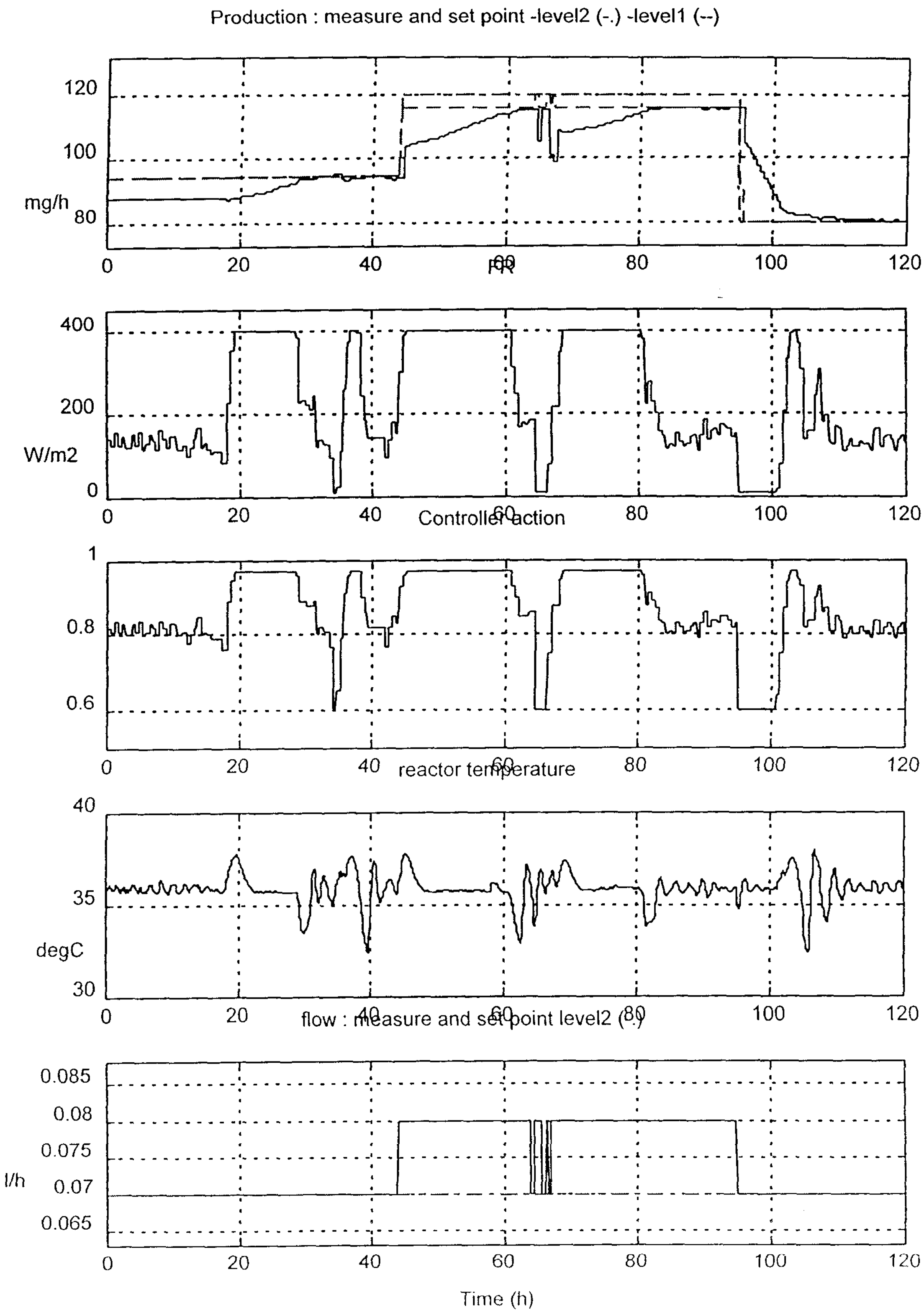


Figure 7 : UAB data Oct 8th 0h00 to Oct 12th midnight

2.3. Reduction of the standard deviation of the controller action

An attempt of reducing the standard deviation of the controller action was done by increasing the horizon on which the average of the biomass concentration, C_{XA}, is measured. This horizon was 10 minutes in figure 7. It is 30 mn (ie the sampling period) in figure 8. The standard deviation of the controller action is reduced from 1.4 10^{-2} (figure 7) to 1.2 10^{-2} (figure 8). With only on experiment, this reduction is not significant. Nevertheless, it is logical to compute the average on the all sampling period, and not on a part of the sampling period.

At times t = 15 h and t = 85 h (figure 8), sudden variations of the production, due to misfunctionning of the concentration sensor like in figure 7, result is sudden variations of the controller action. The same phenomenon, with less intensity, appears between t = 64 h and t = 80 h.

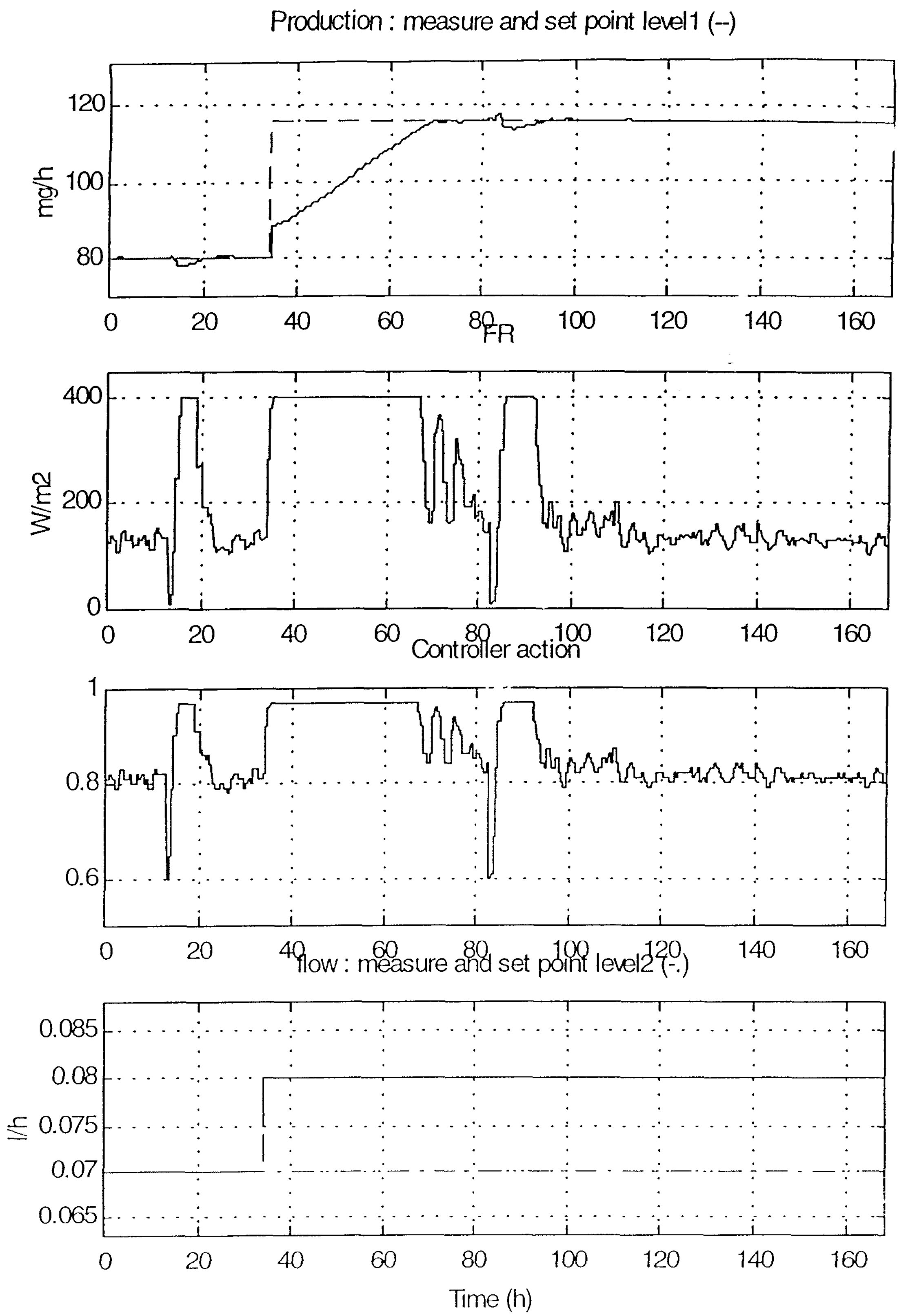


Figure 8 : UAB data from Oct 13th 0h00 to Oct 19th midnight

3. CONCLUSION

The production control of the 7 litres photoautotrophic compartment is now validated at University Autonoma of Barcelona, with no static bias and no overshoot. The step response time at 95 % is about 10 to 15 hours depending on the step value.

This strategy can now be extrapolated to the 70 litres reactor to check that the "first principles" model is generic.

REFERENCES

FULGET N. 1995. MELISSA. Study for the non linear Model Based Predictive Control of Spirulina comportment using knowledge model. ESA contact PRF 142356. Technical Note 24.2.

RICHALET J. 1993. Pratique de la Commande Prédictive. HERMES Edition.

ANNEX 1

Equations of the supervisor

1. INTRODUCTION

The general way followed to cancel a static bias is to put a controller, called supervisor, above the controller of the process as shown in figure A1.1.

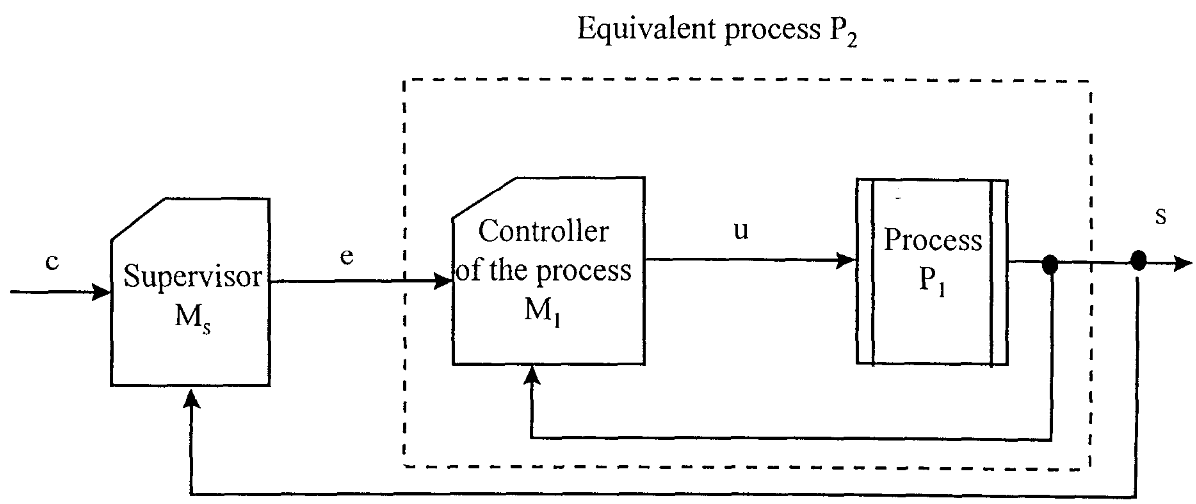


Figure : A1.1 Closed loop system

The process, whose relation between the input u and the output s is called P_1, is the photobioreactor (u is the light intensity F_R and s is the production of biomass).

The internal model of the controller of the process, M_1, is a non linear model extracted from the "first principles" model established by the University of Clermont Ferrand (TN 19.2). The theory of control shows that a static bias is present in that case.

Thanks to the conception of this controller, the relation between the variables e and s, called P_2, can be approximated to a first order transfer :

$$P_2 = \frac{G_2}{1+\tau_2 . p} \qquad (1)$$

with G_2 : static gain
τ_2 : time constant
p : Laplace variable

As P_1 and M_1 are not identical, G_2 is generally different from 1, but not far from 1.

Given trbf, the 95 % closed loop time response of the controller of the process,

$$\tau_2 = \frac{1}{3} \text{trbf}$$

As P_2 is a first order process, it is easily controlled by a first order PFC whose internal model M_s is :

$$M_S = \frac{G_S}{1 + \tau_S p} \qquad (2)$$

with $\left| \begin{array}{l} G_S = 1 \\ \tau_S = \tau_2 \end{array} \right.$

The theory shows that this closed loop system has no static bias.

2. EXPRESSION OF THE OUTPUT OF THE SUPERVISOR

According to (2), the model output at moment n, $s_m(n)$, is

$$s_{m(n)} = \alpha \,.\, s_{m(n-1)} + (1-\alpha)\,.G_s\,.\,e_{(n-1)} \qquad (3)$$

with $\left| \begin{array}{l} \alpha = \exp\left(-\frac{T}{\tau_s}\right) \\ T\text{: sampling period} \end{array} \right.$

At the start of the controller (at n = 0), the variables $s_{m(n-1)}$ and $e_{(n-1)}$ are initialised with the measure of s at that moment, s(0).

$s_{m(n-1)} = s\,(0)$
$e_{(n-1)} = s(0)$

Given

H	:	Horizon of coincidence
λ	:	increment of the first order reference trajectory
c	:	set point

The output of the supervisor, e, is the one of a classical first order PFC (Predictive Functional Control) :

$$e_{(n)} = \frac{\left(c_{(n)} - s_{(n)}\right)(1 - \lambda^H) + s_{m(n)}(1 - \alpha^H)}{G_s(1 - \alpha^H)} \qquad (4)$$

In order not to modify the closed loop time response,

$$\lambda = \alpha \quad (5)$$

so, as $G_s = 1$, (4) becomes

$$e_{(n)} = c_{(n)} - s_{(n)} + s_{m(n)} \qquad \forall n \quad (6)$$

Which is the expression sought after.

3. CONSTRAINTS

The input of the process, u, is constrained between u_{min} and u_{max}.

Given :

s_{m1} : the model output of process P_1 for u equal to one of its constraints ($u = u_{min}$ or $u = u_{max}$).

$D = s_{m1\,(n+H)} - s_{m1(n)}$

The extreme variation of e,Δe, between the moments n and n+H (figure A1.2) is such that :

$$\Delta e - D = \Delta e.\lambda^H \quad (7)$$

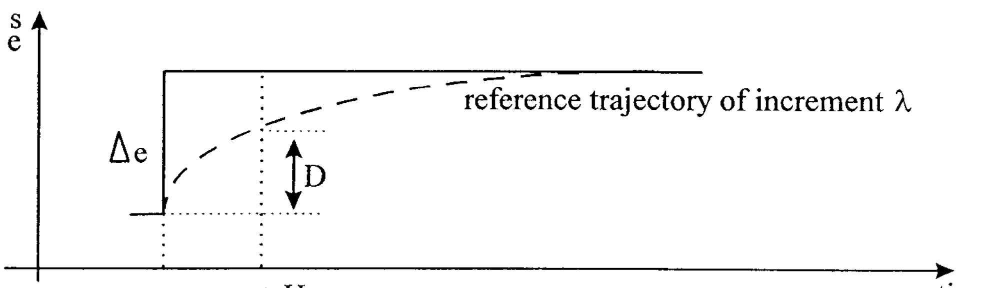


Figure A1.2 : Model output

Given D_{max} and D_{min}, the extreme variations of s_{m1} between the moments n and n+ H, in the cases $u = u_{max}$ and $u = u_{min}$, respectively :

$$D_{max} = s_{m1}\,(n + H,\ u = u_{max}) - s_{m1\,(n)} \quad (8)$$
$$D_{min} = s_{m1}\,(n + H,\ u = u_{min}) - s_{m1\,(n)} \quad (9)$$

The relation (7) gives :

$$\Delta e_{max} = \frac{D_{max}}{1-\lambda^H} \quad (10)$$

$$\Delta e_{min} = \frac{D_{min}}{1-\lambda^H} \quad (11)$$

<u>Connection between the present variables and the variables of the C programme</u>

Present variables	**C programme variables**
D_{max}	dprod_max
D_{min}	dprod_min
λ	lambda
H	nhc
$e_{max} = s_{(n)} + \Delta e_{max}$	cons_prod0_max
$e_{min} = s_{(n)} + \Delta e_{min}$	cons_prod0_min
e	cons_prod0
s	prod

ANNEX 2

C code file with non linear predictive control (V2.2)

```
/*******************************************************************

     NAME        CONTROL.C

     AUTHOR        BINOIS C    (modified by FULGET N.   ADERSA)

     DESCRIPTION
          CONTROL PROGRAM listing file

     UPDATES
          20-09-95

     VERSION : V2.2  (modif. Oct. 1997 by LECLERCQ JJ    ADERSA)
     Modifications according to TN 38.2 :
      1. Elimination of the ligth power controller
         and direct computation of the potentiometer value of the lamps
      2. Elimination of the static bias
      3. Reduction of the standard deviation of the 'controller action'

*******************************************************************/

#include <malloc.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"

int my_interrupt();

/*-------------------------------------
     variables declarations
-------------------------------------*/

VARS   cxa;          /* biomass concentration              */
VARS   nitrate;      /* nitrate concentration              */
VARS   cal_nitrate;  /* nitrate calibration switch         */

VARS   Eb;           /* light intensity in the reactor     */
VARS   Fr;           /* incident flux                      */
VARS   temperature;  /* temperature in the reactor         */
VARS   pH;           /* pH of culture                      */
VARS   act_pompe;    /* dilution pump action               */
VARS   cal_pump;     /* calibration of pump (l/h)          */
VARS   act_lamp;     /* lamp action (dimensionless)        */

/******************* variables ADERSA   V2.2 ***************/

VARS   cons_prod_nom; /* nominal production setpoint       */
VARS   cons_prod_real; /* feasible production setpoint     */
VARS   qe_nom;       /* nominal flow setpoint              */
VARS   qe_real;      /* feasible flow setpoint             */
VARS   production;   /* measured production                */
VARS   prod_mod;     /* model production                   */
```

```
double  sm_sup;        /* model ouput (internal model of the supervisor) */
double  cons_sup;      /* ouput of the supervisor                         */

int     next_pfc;      /* next execution of PFC                  */
char    buffer[100];

/*------------------------------------
      mathematical model
------------------------------------*/

#ifndef ADERSA
model(REACT *react)
#else
model(react)
REACT *react;
#endif
  {

     double  zpc=.135;
     double  zp=.57;
     double  zch=0.0085;
     double  zg=0.;
     double  za;
     double  Ea=871.;
     double  Es=167.;
     double  alpha,delta;
     double  Fr;
     double  R,R1,R2;
     double  z;
     double  jstep=0.01;
     double  pij,pijz;
     double  Kj=20;
     double  KN=5.3;
     double  muM=0.54;
     double  yn=0.42;

     double  coeft,coefN,Rmean;
     double z0, kstep;
     R=0.048;
     R1=0.0302;
     R2=0.02585;

/*  general parameters ---------------------------------*/

     za=zpc+zch;
     alpha=sqrt(za*Ea/(za*Ea+(1+zg)*Es));
     delta=(za*Ea+(1+zg)*Es)*react->Cxa/1000.*alpha*R;

/*  determination of the mean growth rate ---------------*/

     pij=0;
     Fr=react->Fr;
     z0 = 1.e-6 / R;
     kstep = (1. - z0) * jstep;
     for(z=z0;z<1.;z+=kstep)
          {
          if((z<R2/R)||(z>R1/R))
```

```
                {
                pijz=Fr/z*2*cosh(delta*z)/(cosh(delta)+alpha*sinh(delta));
                if(pijz>=1.)
                    {
                    pij+=z*pijz/(Kj+pijz);
                    }
                }
            }
        Rmean=2.*kstep*muM*pij*zpc*react->Cxa*VOLUME_LIGHT/VOLUME_TOTAL;

    /* temperature and nitrates correction */
    /*
        coeft=0.8*exp(-pow((react->temp-35)/10,2))+0.2;
        coefN=react->Cno3/(KN+react->Cno3);
    */


    /****** nitrate saturation ********/
    coefN=1;
    /*************************************/
    /****** no temp correction ********/
    coeft=1;
    /*************************************/
        react->rxa=Rmean*coefN*coeft;
        react->rn=yn*react->rxa;


        }


    /*----------------------------------------------------------
        light calibration from UAB data on 16th September 1997
    ----------------------------------------------------------*/

    #ifndef ADERSA
    lightcal(REACT *react, int mode)
    #else
    lightcal(react, mode)
    REACT *react;
    int mode;
    #endif
      {
        double a_lamp = 6.27;
        double b_lamp = -33.03;

        double Fr;

        switch(mode) {

            case CAL_FR:
            {
            /* FR determination ------------------------*/
            Fr = exp(a_lamp * act_lamp.value) + b_lamp;
            Fr = max(0.,Fr);
            react->Fr = Fr;
            break;
            }
```

```
        case CAL_ACT:
        {
        /*  light controller action determination ----*/
        act_lamp.sp = (log(react->Fr - b_lamp)) / a_lamp;
        break;
        }

        default:
        {
        display_error("Error in routine lightcal ...\n");
        display_error("*** Program terminated ***\n");
        exit(0);
        }
    }

  }


/*-------------------------------------
    copy structure reacta to reactb
-------------------------------------*/

#ifndef ADERSA
copy_react(REACT *reacta,REACT *reactb)
#else
copy_react(reacta,reactb)
REACT *reacta;
REACT *reactb;
#endif

    {
    reactb->Cxa=reacta->Cxa;
    reactb->Cno3=reacta->Cno3;
    reactb->temp=reacta->temp;
    reactb->press=reacta->press;

    reactb->Eb=reacta->Eb;
    reactb->Fr=reacta->Fr;
    reactb->rxa=reacta->rxa;
    reactb->rn=reacta->rn;
    reactb->ro2=reacta->ro2;
    }


/*-------------------------------------
    variables initialisation
-------------------------------------*/

init_vars()
    {
    REACT   init_react;
    double  delta;
    double  prod;
    int     jj;

    display_status("Initialisation of variables ...");
```

```
/* TAG and COMMAND name initialisation     */

        sprintf(cxa.name,"LOOP0107");
        sprintf(nitrate.name,"LOOP0103");
        sprintf(cal_nitrate.name,"DI--0125");

        sprintf(Eb.name,"LOOP0105");
        sprintf(Fr.name,"LOC-0128");
        sprintf(temperature.name,"LOOP0106");
        sprintf(pH.name,"LOOP0104");
        sprintf(act_pompe.name,"LOC-0154");
        sprintf(cal_pump.name,"LOC-0137");
        sprintf(act_lamp.name,"LOC-0146");

        sprintf(cons_prod_nom.name,"LOC-0150");
        sprintf(cons_prod_real.name,"LOC-0152");
        sprintf(qe_nom.name,"LOC-0151");
        sprintf(qe_real.name,"LOC-0153");
        sprintf(production.name,"LOC-0155");
        sprintf(prod_mod.name,"LOC-0156");

/* Variables initialisation  */

        acq_vars();

        init_react.Cxa=cxa.value;
        init_react.Cno3=nitrate.value;
        init_react.temp=temperature.value;
        init_react.Eb=Eb.value;
        lightcal(&init_react,CAL_FR);
        Fr.sp=init_react.Fr;
        write_var(&Fr);
        fill_struct_var(&cxa);

        cons_prod_real.sp=cons_prod_nom.value;
        write_var(&cons_prod_real);
        qe_real.sp=qe_nom.value;
        write_var(&qe_real);

/* Initialisation of the supervisor (for removal of the bias)  V2.2*/
        prod = cxa.value * qe_nom.value;
        sm_sup = prod;
        cons_sup = prod;

/* initialisation timer PFC */
        next_pfc=DT;

        wait_time(1);

        display_status(" ");
        }

/*-----------------------------------
        variables acquisition
-----------------------------------*/
```

```
acq_vars()
	{
	display_status("Acquisition of variables ...");

	read_var(&cxa);

/*  nitrate analyser calibration  */

	read_var(&cal_nitrate);
	if(!cal_nitrate.value)
		{
		read_var(&nitrate);
		}

	read_var(&Eb);
	read_var(&Fr);
	read_var(&temperature);
	read_var(&pH);
	read_var(&act_pompe);
	read_var(&cal_pump);
	read_var(&act_lamp);

	read_var(&cons_prod_nom);
	read_var(&cons_prod_real);
	read_var(&qe_nom);
	read_var(&qe_real);
	read_var(&production);
	read_var(&prod_mod);

	display_status(" ");
	}

/*-----------------------------------
	commands updating
-----------------------------------*/

send_vars()
	{
	display_status("Updating variables ...");

	write_var(&Eb);
	write_var(&Fr);
	write_var(&act_pompe);
	write_var(&cons_prod_real);
	write_var(&qe_real);
	write_var(&production);
	write_var(&prod_mod);
	write_var(&act_lamp);

	display_status(" ");
	}

/*-----------------------------------
	prepare result of control for display
-----------------------------------*/

result()
```

```
{
#ifndef ADERSA
	void display_result(char *,short,short);
#else
	void display_result();
#endif
	char buffer[150];

	sprintf(buffer,"Concentrations  Biomass");
	display_result(buffer,1,1);
	sprintf(buffer,"mg/l");
	display_result(buffer,32,1);
	display_result(buffer,32,2);
	sprintf(buffer,"Nitrate");
	display_result(buffer,17,2);
	sprintf(buffer,"%.1f",cxa.value);
	display_result(buffer,26,1);
	sprintf(buffer,"%.1f",nitrate.value);
	display_result(buffer,26,2);

	sprintf(buffer,"Light");
	display_result(buffer,1,4);
	sprintf(buffer,"Eb      W/m2");
	display_result(buffer,22,4);
	sprintf(buffer,"Fr      W/m2");
	display_result(buffer,22,5);
	sprintf(buffer,"%.1f",Eb.sp);
	display_result(buffer,26,4);
	sprintf(buffer,"%.1f",Fr.sp);
	display_result(buffer,26,5);

	sprintf(buffer,"Production    measured      mg/h");
	display_result(buffer,1,7);
	sprintf(buffer,"%.2f",production.sp);
	display_result(buffer,26,7);

	sprintf(buffer,"set-point      mg/h");
	display_result(buffer,15,8);
	sprintf(buffer,"%.2f",cons_prod_nom.value);
	display_result(buffer,26,8);

	sprintf(buffer,"realised      mg/h");
	display_result(buffer,16,9);
	sprintf(buffer,"%.2f",cons_prod_real.sp);
	display_result(buffer,26,9);

	sprintf(buffer,"model      mg/h");
	display_result(buffer,15,10);
	sprintf(buffer,"%.2f",prod_mod.sp);
	display_result(buffer,26,10);

	sprintf(buffer,"Flow        realised       l/h");
	display_result(buffer,1,11);
	sprintf(buffer,"%.3f",qe_real.sp);
	display_result(buffer,26,11);

	sprintf(buffer,"set point      l/h");
```

```
    display_result(buffer,15,12);
    sprintf(buffer,"%.3f",qe_real.value);
    display_result(buffer,26,12);

    sprintf(buffer,"Next control in    minutes");
    display_result(buffer,45,5);
    sprintf(buffer,"%02d",next_pfc);
    display_result(buffer,61,5);

    }


/*-------------------------------------
    calculate the delta count during time t in minutes
-------------------------------------*/

#ifndef ADERSA
double diff_cpt(VARS *diff_var, int diff_time)
#else
double diff_cpt(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
    {
    int j;
    int i_samp,i_prev,nb_samp;
    double total_count;

    total_count=0;
    nb_samp=ceil(diff_time*60/TSAMP);
    for(j=0;j<nb_samp;j++)
        {
        i_samp=(diff_var->i-j)&NB_SAMP;
        i_prev=(i_samp-1)&NB_SAMP;
        total_count+= ( diff_var->val[i_samp]>=diff_var->val[i_prev]) ?
        diff_var->val[i_samp]-diff_var->val[i_prev] : diff_var->val[i_samp];
        }
    return(total_count);
    }


/*-------------------------------------
    calculate the variable variation during time t in minutes
-------------------------------------*/

#ifndef ADERSA
double diff_var(VARS *diff_var, int diff_time)
#else
double diff_var(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
    {
    double dvar_dt;
    int nb_samp;

    nb_samp=ceil(diff_time*60/TSAMP);
```

```
	dvar_dt=diff_var->val[diff_var->i]-diff_var->val[(diff_var->i
	-nb_samp)&NB_SAMP];
	return(dvar_dt);
	}

/*------------------------------------
	calculate the average during time t in minutes
------------------------------------*/

#ifndef ADERSA
double average_var(VARS *diff_var, int diff_time)
#else
double average_var(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
	{
	int j;
	int i_samp,nb_samp;
	double average;

	average=0;
	nb_samp=ceil(diff_time*60/TSAMP);
	for(j=0;j<nb_samp;j++)
		{
		i_samp=(diff_var->i-j)&NB_SAMP;
		average+=diff_var->val[i_samp];
		}
	average/=nb_samp;
	return(average);
	}

/*------------------------------------
	calculate the average^2 during time t in minutes
------------------------------------*/

#ifndef ADERSA
double average2_var(VARS *diff_var, int diff_time)
#else
double average2_var(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
	{
	int j;
	int i_samp,nb_samp;
	double average;

	average=0;
	nb_samp=ceil(diff_time*60/TSAMP);
	for(j=0;j<nb_samp;j++)
		{
		i_samp=(diff_var->i-j)&NB_SAMP;
		average+=pow(diff_var->val[i_samp],2);
		}
	average/=nb_samp;
	return(average);
```

```
    }

/*------------------------------------
    fill val[i] with the current value
------------------------------------*/

#ifndef ADERSA
fill_struct_var(VARS *fill_struct)
#else
fill_struct_var(fill_struct)
VARS *fill_struct;
#endif

    {
    int jj;

    for(jj=0;jj<=NB_SAMP;jj++)
        {
        fill_struct->val[jj]=fill_struct->value;
        }
    }


/*------------------------------------
    fill val[i] with the current value and delta between each value
------------------------------------*/

#ifndef ADERSA
fill_struct_cpt(VARS *fill_struct,double _delta)
#else
fill_struct_cpt(fill_struct,_delta)
VARS *fill_struct;
double _delta;
#endif

    {
    int jj,kk,ll;

    for(jj=0;jj<NB_SAMP;jj++)
        {
        kk=(fill_struct->i-jj)&NB_SAMP;
        ll=(kk-1)&NB_SAMP;
        fill_struct->val[ll]=fill_struct->val[kk]+_delta;
        }
    }


/*------------------------------------
    calculate the slope of variable by the least mean square method
------------------------------------*/

#ifndef ADERSA
double slope_var(VARS *slope_var,int diff_time)
#else
double slope_var(slope_var,diff_time)
VARS *slope_var;
int diff_time;
```

```
#endif
    {
    int ii,jj,kk;
    int nb_samp;
    double slope;
    double sumxi, sumyi, sumxiyi, sumxi2;

    sumxi=0;
    sumyi=0;
    sumxiyi=0;
    sumxi2=0;

    nb_samp=ceil(diff_time*60/TSAMP);
    for(ii=0;ii<nb_samp;ii++)
        {
        jj=slope_var->i-ii;
        kk=(slope_var->i-ii)&NB_SAMP;
        sumxi+=jj;
        sumyi+=slope_var->val[kk];
        sumxiyi+=jj*slope_var->val[kk];
        sumxi2+=pow((double)jj,2);
        }
    slope=nb_samp*(nb_samp*sumxiyi-sumxi*sumyi)/(nb_samp*sumxi2-sumxi*sumxi);
    return(slope);
    }


/*-------------------------------------
    calculate the slope of counter by the least mean square method
-------------------------------------*/

#ifndef ADERSA
double slope_cpt(VARS *slope_cpt,int diff_time)
#else
double slope_cpt(slope_cpt,diff_time)
VARS *slope_cpt;
int diff_time;
#endif
    {
    int ii,jj,kk,ll;
    int nb_samp;
    double slope;
    double sumxi, sumyi, sumxiyi, sumxi2;
    double raz_cpt;

    raz_cpt=0;
    sumxi=0;
    sumyi=0;
    sumxiyi=0;
    sumxi2=0;

    nb_samp=ceil(diff_time*60/TSAMP);
    for(ii=0;ii<nb_samp;ii++)
        {
        jj=slope_cpt->i-ii;
        kk=(slope_cpt->i-ii)&NB_SAMP;
        ll=(kk-1)&NB_SAMP;
```

```
            sumxi+=jj;
            sumyi+=(slope_cpt->val[kk]-raz_cpt);
            sumxiyi+=jj*(slope_cpt->val[kk]-raz_cpt);
            sumxi2+=pow((double)jj,2);
            if(slope_cpt->val[ll]>slope_cpt->val[kk])
                  {
                  raz_cpt=slope_cpt->val[ll];
                  }
            }
      slope=nb_samp*(nb_samp*sumxiyi-sumxi*sumyi)/(nb_samp*sumxi2-sumxi*sumxi);
      return(slope);
      }


      /**************************************************
      Sign
      ***************************************************/
#ifndef ADERSA
 signe(double x)
#else
 signe(x)
double x;
#endif
      {
      x=(x<0) ? -1 : 1;
      return(x);
      }



/*-----------------------------------
      mathematical model for ADERSA
-----------------------------------*/

#ifndef ADERSA
double predimod(REACT react, double dil, int horiz)
#else
double predimod(react, dil, horiz)
REACT react;
double dil;
int horiz;
#endif
      {
/* react: current reactor model state   (Cxa,Fr,Cno3,temp)       */
/* dil  : dilution rate                 (in h-1)                 */
/* horiz: prediction horizon            (in number of DT)        */
/* prod : predicted production          (in mg/h)                */

      double v, prod;
      int k;

      v  = react.Cxa;      /* current biomass concentration */

      /* model integration (sampling period 1mn) */
      for(k=1;k<=horiz*DT;k++)
```

```
        {
        double Delta;

        react.Cxa=v;
        model(&react);
        Delta=1/60.0*(react.rxa-dil*v);
        v+=Delta;

        }

        prod=v*dil*VOLUME_TOTAL;
        return(prod);
}



/*------------------------------------
     control programm    V2.2
-------------------------------------*/

void control_spiru()
     {

     double  Fr1, Fr2, delfr;                    /*in W/m2 */
     double  prod_ref,prod1,prod2,prod_max,prod_min;      /*in mg/h */
     double  qe_max, qe_min;                      /*in l/h */
     double  dil;                          /*in h-1 */
     double  cxa_moy , nit_moy;                     /*in mg/l */
     REACT   react;
     double   cons_prod0 , cons_prod0_min , cons_prod0_max;
     double   dprod_min , dprod_max;

     acq_vars();

     display_status("Control running ...");

     if(!(next_pfc--))
          {
          /* control PFC algorithm */

          /* biomass concentration */
          cxa_moy=average_var(&cxa,30);
          nit_moy=average_var(&nitrate,10);

          /* production calculation  */
          production.sp=cxa_moy*qe_real.value;

          /* reactor state */
          react.Cno3=nit_moy;
          react.temp=temperature.value;
          react.Cxa=cxa_moy;

          /* flow and production constraints*/
          qe_max=qe_nom.value*(1+DQ);
          qe_min=qe_nom.value*(1-DQ);
          prod_max=qe_max*CXA_MAX;
```

```
prod_min=qe_min*CXA_MIN;

/* feasible production setpoint calculation*/
cons_prod_real.sp=max(prod_min,min(prod_max,cons_prod_nom.value));

/* real flow setpoint  and corresponding dilution rate*/
qe_real.sp=qe_nom.value;
if(cons_prod_real.sp/CXA_MAX>qe_nom.value)
      {qe_real.sp=min(qe_max,cons_prod_nom.value/CXA_MAX);}
if(cons_prod_real.sp/CXA_MIN<qe_nom.value)
      {qe_real.sp=max(qe_min,cons_prod_nom.value/CXA_MIN);}
dil=qe_real.sp/VOLUME_TOTAL;

/* supervisor */
sm_sup = LAMBDA * sm_sup + (1. - LAMBDA) * cons_sup;
/* 1_ output of the supervisor */
cons_prod0 = cons_prod_real.sp - production.sp + sm_sup;
/* 2_ carrying forward the absolute constraints FR_MIN FR_MAX */
react.Fr = FR_MAX;
dprod_max = predimod(react,dil,NHC) - production.sp;
cons_prod0_max = production.sp + dprod_max/(1.-pow(LAMBDA,NHC));
react.Fr = FR_MIN;
dprod_min = predimod(react,dil,NHC) - production.sp;
cons_prod0_min = production.sp + dprod_min/(1.-pow(LAMBDA,NHC));
cons_prod0 = max(cons_prod0_min,min(cons_prod0_max,cons_prod0));
cons_sup = cons_prod0;

/* reference trajectory */
prod_ref=cons_prod0-pow(LAMBDA,NHC)*(cons_prod0-production.sp);

/*first scenario */
Fr1=Fr.value;
react.Fr = Fr1;
prod1=predimod(react,dil,NHC);

/* second scenario */
delfr=DFR*signe(cons_prod0-production.sp);
Fr2=Fr1+delfr;
react.Fr = Fr2;
prod2=predimod(react,dil,NHC);

/* Fr calculation */
Fr.sp=Fr.value+(prod_ref-prod1)/(prod2-prod1)*delfr;

/* constraints on Fr */
Fr.sp=max(FR_MIN,min(FR_MAX,Fr.sp));

/* light action sended to output of P100 controller */
react.Fr=Fr.sp;
lightcal(&react,CAL_ACT);

/* pump setpoint sended to P100 controller */
act_pompe.sp=qe_real.sp/cal_pump.value;

/* model output calculation */
prod_mod.sp = predimod(react,dil,1);
```

```
            next_pfc=DT;
            }

        send_vars();
        result();
        }
□
```