Universitat Autònoma de Barcelona
Dep. Enginyeria Química
08193 Bellaterra, Barcelona, Spain

MELISSA

Memorandum of Understanding
ECT/FG/MMM/97.012

Contract Number: ESTEC/CONTRACT11549/95/NL/FG

Technical Note: 43.7

General Purpose Station 98

Version: 1
Issue : 0

MENGUAL X.; ALBIOL, J.; GODIA, F.

July 2000

Document Change Log

Version	Issue	Date	Observations
Draft	0	10/03/98	Preliminary Version
1.0	1	7/7/2000	Final Version

Table of Contents

1 INTRODUCTION

Efficient and reliable operation of the MELISSA pilot plant, depends either on the efficient operation of each of its sub-units or compartments, as well as on the decisions taken at a higher level of control in order to coordinate the operation of the different units and assuring crew survival. This higher level of control is still under development, however it is already linked of the operational control of several units of the pilot plant

The scale up of the bioreactor for Compartment IV to a new volume of 77 litres, performed during the previous phase of the pilot plant operation, implied the installation of new low level controllers (AC20) for its operation. For this reason it was necessary to incorporate them inside the data monitoring control and data storage network of the pilot plant. In this phase they have been incorporated inside the general control scheme of the pilot plant, with the required software upgrade.

The general purpose station (G.P.S.) v2.3 is basically the adaptation of the previous version of the G.P.S., which was used to control the operation the 7 litres air-lift reactor of compartment IV with *Spirulina platensis*, to the new 77 litres bioreactor. For this upgrade, all the parameters used to describe the physical characteristics of the new bioreactor were redefined with the new values.

In order to improve the user interface, allowing for the modification and visualisation of new parameters and variables, new screens were introduced in G.P.S. for the particularities of Compartment IV.

After this first adaptation, installation and testing of the software, two more upgrades, related with the Compartment IV control routines were also implemented. The first upgrade, involves the renaming of the high level control software, the name of its control routines and of the Compartment IV control routine.

2 MODIFICATIONS DUE TO THE UPGRADE OF THE AIRLIFT LOOP BIOREACTOR

As a first step in the upgrade of the new software, a new version of the controller's interface, Toptools from Industar, was purchased and installed in the computer stations. The new version included the low-level control interfaces with the

new ASCON AC-20 controllers. Proper use of this software interface implied several steps such as definition of new variables in the Toptools software, for the new controllers, definition of new functions in the GPS to access this variables, and modification of the control subroutines for reading from the new variables, instead of from the previous ones. After those steps the software was validated in operational conditions.

2.1 The New Communications Functions

Due to the inclusion of the new low level controllers, which responded to a different communication protocol and numerical data format, it was necessary to write new communication functions in the GPS allowing to communicate with the INDUSTAR program Toptools V3.6. The newly created functions were named *rd_ascon* and *wr_ascon* respectively to read and write. They can only operate with the new ASCON AC20 controllers. The variables monitored and controlled by the ASCON AC20 correspond to the ones defining the operational state of the new 75 litres airlift loop bioreactor. However the previous communication functions *write_var* and *read_var* were conserved to allow for the communication with the MICON controllers still in operation, used in the other compartments. The two new functions were included in the module *Ctrlspiru.c.*

RD_ASCON

This is the function that will be used to read the variables from the INDUSTAR Tooptols software, representing the values measured by the ASCON AC20 controllers. This function is based in the procedure *rd_gps*, which is given by INDUSTAR´s libraries, and corresponds to the low level read functions from INDUSTAR software. The source code of these functions has been included in section 5 for further reference.

WR_ASCON

This function was created to allow the data writing on Industar software, and was named *wr_ascon*. This function writes from the G.P.S. to the variables referred in INDUSTAR software for the ASCON AC20 controllers.

This function is based on the function *wrm_gps*, which is also given by INDUSTAR´s libraries. The source code of these functions has also been included in section 5 for further reference.

2.2 New Variables

In order to upgrade Compartment IV from the 7 litres air-lift bioreactor to the new 75 litres airlift loop bioreactor, it was required to define new variables inside INDUSTAR Tooptols V3.6 software to work with the ASCON AC20 controllers.

New variable name	Variable used for	Previous variable name name
LOC5431	Set Point Production	LOC-150
LOC5433	Set Point Flow	LOC-151
LOC5435	Feasible Production	LOC-152
LOC5437	Feasible Flow	LOC-153
LOC5439	Measured Production	LOC-155
LOC5441	Measured Flow	
LOC5443	Biomass	LOOP0107
LOC5445	Radiant Flux (Fr)	LOC-128
LOC5447	Output Pump	LOC-154
LOC5449	Calibration Pump	LOC-137
LOC5451	Model Biomass Production	LOC-156
LOC5453	Temperature	LOOP0106
LOC5455	pH	LOOP0104
LOC5457	Eb	LOOP0105
LOC5459	Nitrate	LOOP0103
LOC5461	Light Regulation	LOC-112
LOC5463	Measured Production	
LOC5465	Balance 1	
LOC5467	Balance 2	
LOC5469	Balance Actuation	

Table 2.2.1 : New variables declared for new 75 litres bioreactor

Once the new Toptools software was installed, new variables and screens were defined, allowing for a first step in the data visualisation and recording. This software makes available the controller's data for the control processes taking place in the G.P.S. The new variables included in the Toptools database operated, in a first step, correctly and the G.P.S. was able to access all the controller's data defined in first place. However, access problems appeared in the variables that previously allowed its access correctly, as new variables were added to the database. Basically the problem was that the variables (mainly LOC values) allowed the G.P.S. to modify its values, but not to read them. For this reason it was decided to include new screens in the GPS, to allow the specification of set points directly on the GPS, providing an alternative to the use of Toptools as a user interface for the input of the set points.

The new variables are the ones where the G.P.S. read and write when it operates as a higher level of control for the operation of the bioreactor. The new variables defined are summarised in table 2.2.1.

The new variables make reference to new memory positions in the ASCON AC20 controllers while the previous ones referred to memory addresses of the previous MICON controllers.

2.3 New reactor parameters

As mentioned previously, it was necessary to modify the values of the parameters describing the physical characteristics of the new bioreactor. These are key values used by the control system to perform its control task. All this variables are defined in the 'include' header file *Melissa.h*. The modified values are presented in table 2.3.1.

Constant Name	Description	Previous values 7 l. Bioreactor	New values for 75 l. bioreactor
VOLUME_LIGHT	Illuminated volume	3.9 litres	53 litres
VOLUME_TOTAL	Volume of reactor	7 litres	77 litres

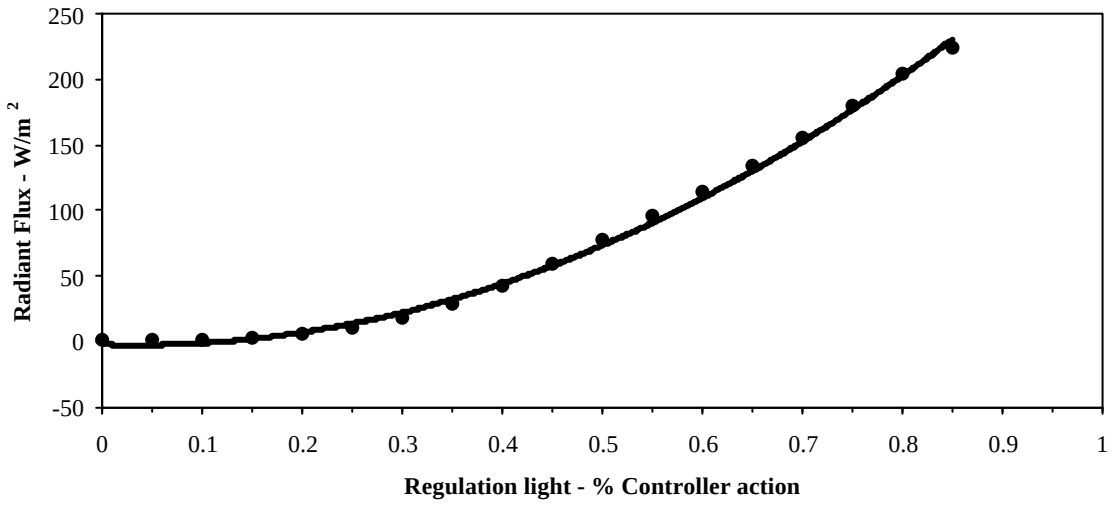
Table 2.3.1 : New constants values for the new 75 litres bioreactor

2.4 New Radiant Flux(Fr) and Light Regulation calculation

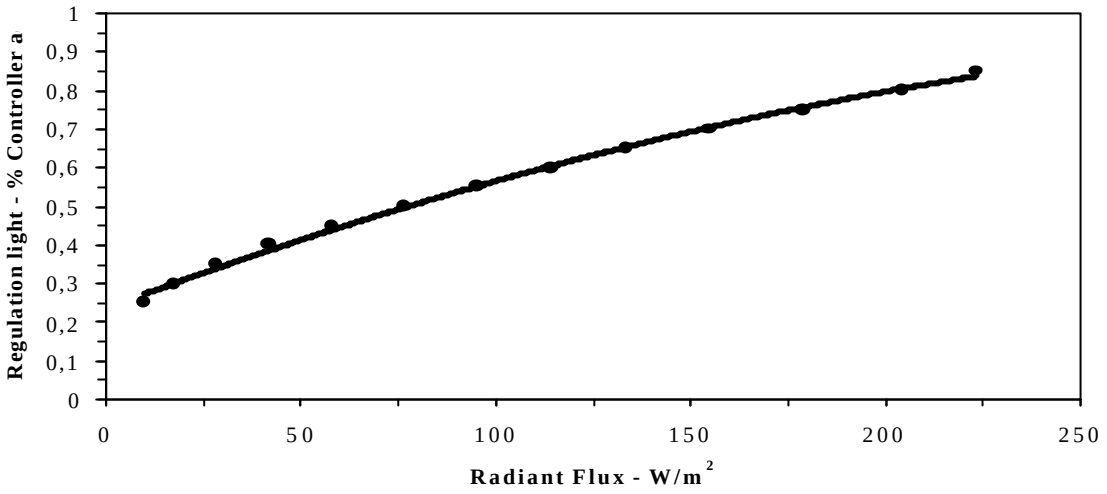
With the new 75 litres bioreactor it was necessary to update the functions used for the conversion of the controller action to the light intensity or radiant flux at the surface of the bioreactor (F_0 (W/m^2)). It was also necessary to define a function for the reverse operation, that is for the calculation of the output light regulation (%Controller action) from the F_0. In table 2.4.1 it can be found the new expressions developed for the calculation of new parameters, and in the figures. 2.4.2 and 2.4.3 a graphical representation of the new functions used can be found.

	Previous expressions for 7 litres reactor	**New expressions for 75 litres bioreactor**
Radiant Flux (Fr) (W/m2)	Fr=e(6.27· r)-33.03	Fr=350.93· r2 - 23.858· r - 2.256
Light Regulation (r) (%Controller action)	$r = \frac{\log(Fr + 33.03)}{6.27}$	r =-5 10-6 Fr2+0.038· Fr+0.3259

Table 2.4.1: New equations expressing the relationship between radiant flux at the reactor surface and controller action.



Graph 2.4.2: Radiant Flux (W/m^2) calculation: y=350.93· x^2-23.858x-2.256



Graph 2.2: Regulation Light (% Controller action) calculation: $r=-5\ 10^{-6}\cdot Fr^2+0.038\cdot Fr+0.3259$

As can be seen in the figure 2.2, the calculation of the light regulation value is valid only in the interval [9.5, 225] of the radiant flux, therefore a limit for the working range was introduced inside the G.P.S subroutine for this calculation. If a conversion value outside this range is required, the function returns the value of the nearest limit. As an example, if the conversion of light intensity to the controller action gives a value over the maximum control value of 85% for light regulation, the maximum of 0.85 is returned. Alternatively a value under the minimum is required, the lower value of the range (0.25) is returned.

3 MANUAL INPUT OF VALUES

3.1 New Input Values on the GPS

As mentioned before, for the proper operation of Compartment IV, some parameters required by the software have to be changed by the operator. To introduce those values a new screen inside G.P.S. was designed which will be accessed by the "Shift +F1" combination keys. The new screen has been defined within the module *Results.c* of G.P.S source code software. An example of the screen for the manual input is shown in figure 3.1 based on a screen captured from the operating G.P.S. In the following the values that the user can modify will be described.

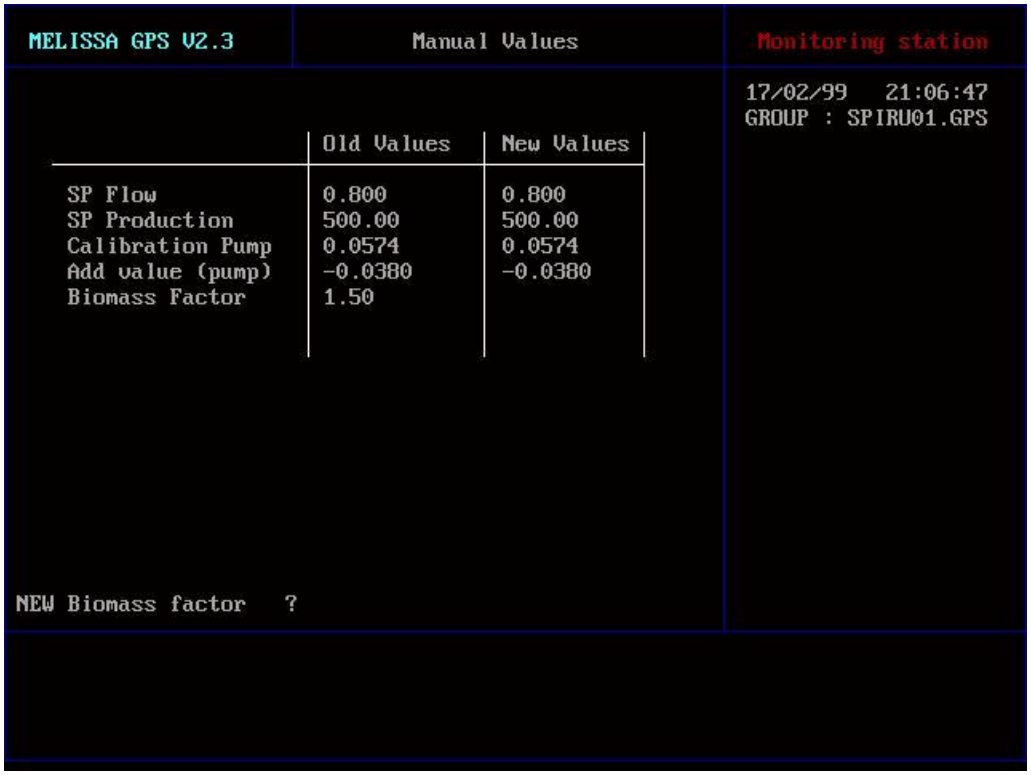


Fig. 3.1 Example of the new screen for introduce manual values

3.1.1 Set-point production and Set point flow

The compartment IV control routines require two set-points, these are the productivity set-point and the liquid flow set-point. The control routines act on the manipulated variables of the bioreactor, in order to attain these values. These two values can be changed in the new parameter values input screen.

3.1.2 Biomass factor

The controllers of Compartment IV can only read the absorbance of the bioreactor, but the control model needs the value of biomass dry weight to calculate the productivity. Therefore inside the G.P.S. the biomass concentration is calculated as follows:

$$\text{Biomass dry weight} = (\text{absorbance}) \cdot (\text{biomass factor})$$

The biomass factor is the value that can be changed in the input screen to allow this calculation.

3.1.3 Calibration parameters for the liquid flow pumps (litres/hour.)

The controllers of Compartment IV operate on the liquid flow pumps only, through a value of percentage of action of the pump. However the G.P.S. control model requires the real liquid flow. Therefore it is necessary to convert the value of the liquid flow, to the percentage of action inside G.P.S. This conversion can be done with the following equation:

Flow=(%Pump)· (cap)+(adv)

Where the following values are required:

% Pump : percent of pump action.

cap: pump calibration factor.

adv: constant deviation factor.

The last two factor values are the ones that the user can modify to adapt the pump operation to the required flow.

Another modification introduced in the G.P.S and related with the operation of the liquid pumps, was the introduction of the liquid flow regulation using two pumps. This allows to easily choose the input flow source. Each pump obtains its input from a different vessel, and by using the measured weight of the vessel it is possible to identify when one vessel is empty and to use the other one. This is done automatically by the low level controllers. Therefore G.P.S. program code was modified to identify which of the two pumps was operating. The detection method is simple. In order to know which pump is running, the action value of the first pump is monitored. If its value is 0 it is assumed that the pump is off. In that case the action value of the second pump is used. In case its value is also 0, an error message is issued to inform the user that the flow is stopped. Otherwise the action value of the first pump is used.

3.2 Default values for the manual input

In order to have a safety measure in front of power failure, a system to recover the parameter values that are manually introduced, was implemented. In the following, the operation of this system is described.

During the G.P.S. initialisation after power up, the values of the parameters are read from a file named "SpirVal.txt". If this file does not exist, or the G.P.S. can not open it, the system uses some standard default values, defined previously such as the ones described in table 3.2.1:

Default values	Value
Set point production	500
Set point Flow	0.7
Calibration pump	0.0574
Add value for pump	-0.038
Biomass factor	1.5

Table 3.2.1 Default values for the operation parameters of compartment IV.

When one or several of those values are modified by the operator, during the course of the G.P.S. operation, the file "SpirVal.txt" is automatically upgraded in order to save the new values.

4 NEW DEBUG SCREEN FOR COMPARTMENT IV

In order to evaluate the operation of the G.P.S. during runtime it was considered interesting to have a screen were the values of the most important regulation variables, and the actual values of the parameters being used, could be displayed. These values are more often used during de debugging procedure of the G.P.S. However it was decided that it could be interesting to have some of this values during normal operation and therefore they were not removed in the final version of the software. This new screen was defined inside de module *Results.c* of G.P.S. An example of the screen during operation can be found in figure 4.1.

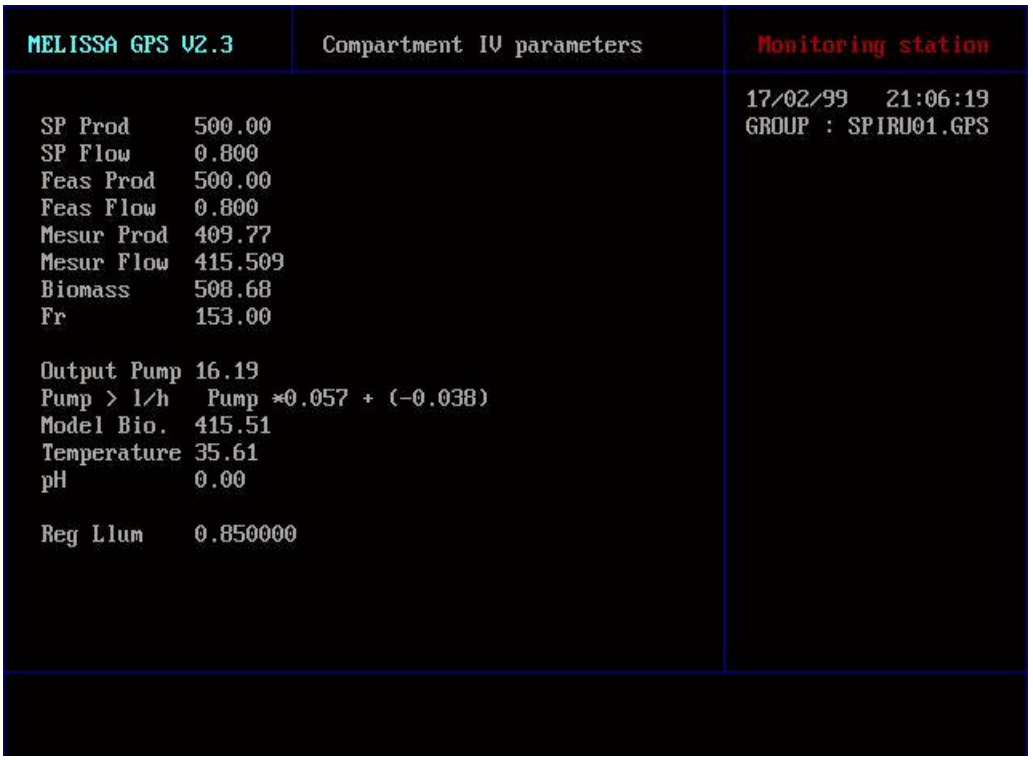


Fig. 4.1 Example of the new screen for debug control of Compartment IV.

5 G.P.S. UPGRADE JUNE-1999

5.1 Introduction

Future upgrades of the control system software, will allow to control several compartments at the same time as well as to coordinate the operation of all of them. Therefore in order to prepare the present version of the control software for those upgrades, several modifications were required. Following the specifications indicated by ADERSA, modifications were implemented in order to improve the independency of control modules, from the main control program. As examples, the change of the names of some software components or the modification of several components of compartment IV control, can be mentioned.

5.2 Upgrade Software Names

According with the results of the ESA/LGCB/ADERSA meeting on May 12th 1999, previous software module names were changed to the new official software names. The new names are summarised in table 6.2.1

NAME	SOFTWARE
LSPC	Light Spirulina Production Control
LRPC	Light Rhodobacter Production Control
SBQ	Spirulina Biomass Quality
MCS	Melissa Control Software (Level 2 of the hierarchical control)

Table 6.2.1 : New official names of control software.

5.3 Upgrade of the High Level Control Software

The high level control software, previously known as the G.P.S. software as it was run at the "General Purpose Station", has been renamed as "Melissa Control Software" (M.C.S.).

5.4 Upgrade of the Spirullina Control Routine

The new routine to perform the control of the biomass production by action on the light intensity will be renamed as 'Light Spirulina Production Control' and abbreviated LSPC .

The communication of this function with the main MCS program is now exclusively done by the use of its arguments, .

All the variables inside the function are local variables and none of them are global variables, common with the main program. The call to the control routine will be:

LSPC (PROD_SP2, CX, QE_SP2, FR, QE_MES, SM_SUP, CONS_SUP,VOL, FI, DT, LAMBDA, INIT, VAR_OUT)

The arguments of this function are described in table 6.4.1

Argument	v/p	DESCRIPTION
PROD_SP2	v	level2 production set point (g/h)
CX	v	biomass concentration (g/l)
QE_SP2	v	level2 flow rate set point (l/h)
FR	p	light intensity : measured or computed by the control (W/m2) . input argument : measured value of FR . output argument : computed by the control
QE_MES	v	measure of flow rate (l/h)
SM_SUP	p	production model output computed by the supervisor (g/h) . input argument : value at previous moment . output argument : value at present moment
CONS_SUP	p	production set point computed by the supervisor (g/h) . input argument : value at previous moment . output argument : value at present moment
VOL	v	volume of the reactor (l)
FI	v	illuminated surface fraction (no dimension)
DT	v	control period (h)
LAMBDA	v	Dynamic of the reference trajectory (dimension less)
INIT	p	initialisation flag (when equal to 0) put to 1 by this programme
VAR_OUT[0]	p	level 1 production set point (g/h)
VAR_OUT[1]	p	level 1 flow rate set point (l/h)
VAR_OUT[2]	p	model output of the Spirulina growth at next instant (l/h)

Table 6.4.1 : Arguments of the function lspc (v: numerical variable, p: pointer)

5.4.1 Modification of the function *control_spiru*

The call to the subroutine LSPC is done by the function *control_spiru* located at the *'ctrlspir.c'* source file and replaces all the calculations directly connected to the control.

In the following, the previous and new versions of function *control_spiru* are listed:

5.4.1.1 Previous function *control*_spiru

```
/*-------------------------------------
       control programm
-------------------------------------*/

int control_spiru(ScreenViews active_reactor)
       {
       FILE *fp;
       int retval = 0;
       char buffer[300], buffer2[300];
       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;

    sprintf (buffer, "Accessing spiruline data ...");
    _outtext (buffer);

    acq_vars_spirulina();

    #ifndef __Monitoring
    display_status("Control running ...");
    #endif

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

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

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

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

                        /* second scenario */
                        delfr=DFR*signe(cons_prod_real.sp-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_sp=DT;
                        }
                send_vars_spirulina();

            sprintf (buffer, " done\n");
            _outtext (buffer);

                return retval;
                }
```

5.4.1.2 New function *control_spiru*

```
/*-------------------------------------
        control programm
-------------------------------------*/

int control_spiru(ScreenViews active_reactor)
        {
        FILE *fp;
        int retval = 0;
        char buffer[300], buffer2[300];
        REACT   react;
        double  cx;                             /*in g/l */
        double  dt, fI, var_out[3];

    sprintf (buffer, "Accessing spiruline data ...");
    _outtext (buffer);
    acq_vars_spirulina();
    #ifndef __Monitoring
    display_status("Control running ...");
    #endif
        if(!(next_pfc_sp--))
                {
                /* control PFC algorithm */
                /* mean biomass concentration */
                cx=average_var(&cxa,10) / 1000.;
                /* illuminated surface fraction */
                fI = VOLUME_LIGHT / VOLUME_TOTAL;
                /* control period expressed in h */
                dt = DT / 60.;

                /* light spirulina production control */
                lspc(cons_prod_nom.value, cx, qe_nom.value,
                     &Fr.value, qe_real.value, &sm_sup, &cons_sup,
                     VOLUME_TOTAL, fI, dt, LAMBDA, &init_spir, var_out);

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

                /* real flow rate set point */
                qe_real.sp = var_out[1];
                /* pump setpoint sended to P100 controller */
                act_pompe.sp=qe_real.sp/cal_pump.value;

                /* model output of spirulina growth */
                prod_mod.sp = var_out[2];

                /* feasible production setpoint */
                cons_prod_real.sp = var_out[0];

                next_pfc_sp=DT;
                retval = 1;
                }
        send_vars_spirulina();
```

```
    sprintf (buffer, " done\n");
    _outtext (buffer);
        return retval;
        }
```

5.4.2 Modification of function *init_vars_spirulina*

A few instructions of the function *init_vars_spirulina* were also changed :

Previous instructions

```
/* Initialisation of the supervisor (for removal of the bias) V2.2 */

       prod = cxa.value * qe_nom.value;

       sm_sup = prod;

       cons_sup = prod;
```

New instruction

```
/* Initialisation of the supervisor (for removal of the bias) V3.0 */

       init_spir = 0.;
```

5.4.3 Cancellation of functions

A few previous functions of *ctrlspir.c* that are now included in *lspc.c*, were removed. These functions are :

model (replaced by *dercx* in the source file *lspc.c*);

predimod (replaced by *mod_spir* in the source file *lspc.c*).

NB: The mathematical functions *signe* , *min* and *max* are still necessary. The function of light calibration, *lightcal*, which is considered as an outside part of the control, remains in the source file *ctrlspir.c* so that it could be modified when it is needed for example if a change of equipment occurs.

5.4.4 Modification of *melissa.h* header file.

The following parameters, that are modifiable by the control user in order to adapt the software to the hardware, are defined in the include *melissa.h* :

VOLUME_LIGHT : illuminated reactor volume;

VOLUME_TOTAL : total volume of the reactor;

DT : control period;

LAMBDA : reference trajectory dynamic.

The other parameters, that are modifiable by the engineer in charge of the first principles model or of the control, are embedded in the source file *lspc.c* and had to be removed from *melissa.h* .

Hereafter the new version of *melissa.h* header file is listed:

```
#ifndef __melissa_def
#define __melissa_def

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

       NAME            MELISSA.H

       AUTHOR          BINOIS C   (modified by FULGET N. ADERSA)

       DESCRIPTION
              General Declarations

       UPDATES
              20-09-95

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



/*-------------------------------------
       constants for VARS
-------------------------------------*/
#define TAG     128
#define CMD     255
#define UNDEF   0
#define NB_SAMP 0xFF          /* number of samples stored in val[ ] */


/*-------------------------------------
       structure for variables
-------------------------------------*/

typedef struct _vars {
       char    name[9];        /* tag name             */
       char    file[12];       /* file name            */
       int     type;           /* rd_gps/set_cmd return code            */
       unsigned char tag_cmd;  /* TAG or CMD           */
       unsigned int dev_num;   /* controller number    */
       double  value;          /* current value        */
       int     i;              /* pointer on last value entered in val[] */
       double  val[NB_SAMP+1]; /* previous values      */
       double  min;
       double  max;
       double  sp;             /* set point for LOOP       */
       double  out;            /* out value for LOOP       */
```

```
       char    unit[5];        /* unit for analog values   */
       char    update;         /* ON when structure updated*/
} VARS;

/*-------------------------------------
       structure for opened GPS files
-------------------------------------*/

typedef struct _gps_file{
       char    file[15];           /* file name               */
       int     handler;            /* handler of gps file     */
       int     rank;               /* rank of the gps file    */
       struct  _gps_file   *next;  /* next opened gps file    */
} GPS_FILE;


/*-------------------------------------
       structure for reactor state
-------------------------------------*/

typedef struct  _react{
       double  Cxa;    /* in mg/l      */
       double  Cno3;   /* in mg/l      */
       double  temp;   /* in degre C   */
       double  press;  /* in           */
       double  Eb;     /* in W/m2      */
       double  Fr;     /* in W/m2      */
       double  rxa;    /* in mg/l/h    */
       double  rn;     /* in mg/l/h    */
       double  ro2;    /* in mg/l/h    */
} REACT;

/*-------------------------------------
       general constants
-------------------------------------*/

#define TSAMP_MAIN   60         /* sampling interval in secondes       */
#define SYNCHRO  1
#define ERROR_SPEED 0.01        /* max error for the model      */
#define VOLUME_LIGHT 3.900      /* illuminated volume (in l)    */
#define VOLUME_TOTAL 7.000      /* total volume (in l)          */

/*-------------------------------------
       Model control constants
-------------------------------------*/

#define CAL_FR  10
#define CAL_ACT  20


/*-------------------------------------
       Mathematical constants
-------------------------------------*/

#define PI      3.14159265359


/*-------------------------------------
       colours
-------------------------------------*/
```

```
#define BLACK   0
#define BLUE    1
#define GREEN   2
#define CYAN    3
#define RED     4
#define MAGENTA 5
#define BROWN   6
#define WHITE   7

/* ====================================
       Definitions for displaying
       multiples reactors
====================================== */

typedef enum _reactors { principal,
                        spirulina,
                        gspirulina,
                        igspirulina,
                        nitrifying,
                        gnitrifying,
                        ignitrifying,
                        reactor_3,
                        reactor_4,
                        help_1,
                        status,
                        filemanagement } ScreenViews;

/*------------------------------------
       ADERSA constants
------------------------------------*/

#define DT      30              /* sampling period of PFC       */
#define DT_NITROGEN     30      /* sampling period of PFC       */
#define LAMBDA  0.88            /* reference trajectory dynamic */

extern int SAMP_GLOB;

/* ================================
    output file definitions.
   ================================ */

#define CONFIG_FILE "melissa.cfg"

/*#define __Monitoring    /* Just for read only */

#endif
```

5.4.5 Modificaton of *ctrlspir.h*

The modification of this include file, concerns only the declaration of the global variable *init_spir* .

Its new version is given hereafter.

```
#ifndef __ctrlspir_def
#define __crtlspir_def
/*****************************************************************
```

```
NAME            CTRLSPIR.H

AUTHOR          Pedro Pons

DESCRIPTION     Headers and declarations for the spiruline controler.

UPDATES
       27-05-96

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

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

#include "userdef.h"
#include "melissa.h"
#include "results.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    act_lamp;       /* lamp action (dimensionless)         */
VARS    cal_pump;       /* calibration of pump (l/h)           */

/*******************   variables ADERSA   *****************/

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 output (internal model) of the supervisor */
double  cons_sup;         /* output of the supervisor */
double  init_spir;       /* initialisation flag of spirulina control 'lspc'*/

int     next_pfc_sp;       /* next execution of PFC for spirulina reactor */
char    buffer[100];

/***********************************
       variables defined to get the history of
       the system for the input filters
***********************************/
```

```
extern double ALPHAFILTERcxa;
extern double ALPHAFILTERnitrate;
extern double ALPHAFILTEREb;
extern double ALPHAFILTERpH;

extern int SAMP_SPIRU;

int first_time_spiruline;

#endif
```

6 M.C.S. UPGRADE SEPTEMBER-1999

6.1 Introduction

In this upgrade some software parameters were modified, related with the control of the compartment IV following the last ADERSA study about the Model Based Predictive Control of Spirulina (TN-44.1).

Few parameters of the control , relative to the light supply and to the geometry of the reactor, were also modified in order to be adapted to the new 77 litres reactor.

6.2 UP-DATE of the controller

6.2.1 Geometrical parameters of the 77 litres reactor

In comparison with the 7 litres reactor which consisted in 2 concentric cylinders, the 77 litres reactor is composed of 2 separated and parallel cylinders connected in a loop; the radius of each cylinder is equal to 0.076 m.

6.2.2 Up-date of the control software

The control software is modified in order to take into account the different operation of new light power supply system and the new bioreactor geometry. The modifications are located in 2 subroutines (*lspc* and *dercx* of the source file *lspc.c*) whose previous and new versions are described hereafter.

6.2.2.1 Up-dating of the subroutine 'lspc'

Previous version :

```
double  Fr_max=400.;    /* max constraint on FR (in W/m2) */
```

New version :

```
double  Fr_max=223.;    /* max constraint on FR (in W/m2) */
```

Modifications done :

Modification of the maximum light intensity according to previously given UAB data (TN 37.2 ; April 1998, reported in annex 1).

6.2.2.2 Up-dated subroutine 'dercx'

Previous version :

```
       double  RT=0.048;       /* external radius of the reactor */
double  R1=0.0302;      /* int radius of the external cylinder */
       double  R2=0.02585;;    /* ext radius of the internal cylinder */

       /*
       Computation of rx (biomass growth rate)
       Integration interval : [z0, 1]
       This interval is divided into 'nstep = 1/jstep ' equal parts
       Integration : rectangle method
       */
       z0 = 1.e-6 / RT;
       kstep = (1. - z0) * jstep;
       sX = 0.;
       for (z=z0; z<1.; z+=kstep)
         {
         if(( z<= R2/RT)||(z >= R1/RT)) /* concentric cylinders */
           {
           pijz = 2*Fr/z*cosh(delta*z)/(cosh(delta)+alpha*sinh(delta));
           if (pijz>=Fmin)
             sX += z * pijz / (Kj+pijz);
           }
         }
       rx = 2. * muM * cx * sX * fI * zpc * kstep;
```

New version :

```
       double  RT=0.076;       /* external radius of the reactor */
       /*
       Computation of rx(biomass growth rate)
       Integration interval : [z0, 1]
       This interval is divided into 'nstep = 1/jstep ' equal parts
       Integration : rectangle method
       */
       z0 = 1.e-6 / RT;
       kstep = (1. - z0) * jstep;
       sX = 0.;
       for (z=z0; z<1.; z+=kstep)
         {
         pijz = 2*Fr/z*cosh(delta*z)/(cosh(delta)+alpha*sinh(delta));
         if (pijz>=Fmin)
           sX += z * pijz / (Kj+pijz);
         }
       rx = 2. * muM * cx * sX * fI * zpc * kstep;
```

6.2.3 Radius value modification

The radius `RT` has been set to its new value.Accordingly the instruction :

```
if(( z<= R2/RT)||(z >= R1/RT)) /* concentric cylinders */
```

is removed of the previous version because of the two parallel cylinder vessel configuration of the 77 (instead of 2 concentric cylinders).

6.2.4 Light calibration UP-DATE

Several modifications were done in the functions which calculate the conversion of the Manipulated Variable value (the light intensity, FR, computed by the control *lspc*) into the Action Variable 'light controller action' , which is specific of the light power supply system.

In this module the new equations fitted by ADERSA, following UAB data from (TN37.2,p.22,April 1998), were used. The new expressions are shown in the table 7.3.1

Parameter	**Old function**	**New adjusted functions**
Radiant Flux (W/m^2) (Fr)	$Fr=350.93 \cdot r^2 - 23.858 \cdot r - 2.256$	$Fr=289 \cdot r^2 + 54.56 \cdot r - 24.19$
Regulation Light (r) (%Controller action)	$r = -5 \cdot 10^{-6} \cdot Fr^2 + 0.038 \cdot Fr + 0.3259$	$r = (-54.56 + (109.12 - 1156 * (-24.19 - Fr))^{1/2} / 578$

Table 6.2.4.1 : New adjusted equations for light calibration

6.2.5 Modification of parameter values for the illuminated surface fraction

Following the indications of ADERSA, the calculation of the variable 'illuminated surface fraction' (f_i) has been changed, in the function *control_spiru* using the expression fi=VOLUME_LIGHT/VOLUME_TOTAL , by a constant value defined in *melissa.h.*

7 SUMMARY OF THE CONTROL SYTEM TESTS

After each of the modifications were introduced the control system was thoroughly tested to evaluate its proper operation. In the following, the final control system tests, preformed after all the previous modifications were done, are given. This offers a view of the software performance at present time.

7.1 Production Test

To tests and evaluate the performance of the software in driving the biomass production to the user defined set point, several tests involving variation in the biomass production set point were performed.

7.1.1 Increase in the Production Setpoint

Figure 7.1.1.1 Shows the evolution of the biomass concentration in front of an increase in the biomass production set point. Following its modification, the control system immediately increases the light intensity set point which results in an increase in the light intensity at the surface of the bioreactor (Fr). As a consequence the biomass concentration in the bioreactor increases as expected, resulting in an increased productivity. As biomass productivity approaches the new set point the action in the light control system is decreased, diminishing the Fr value. The biomass concentration ceases to increase and the control system regulates the light intensity to maintain this new productivity set point.

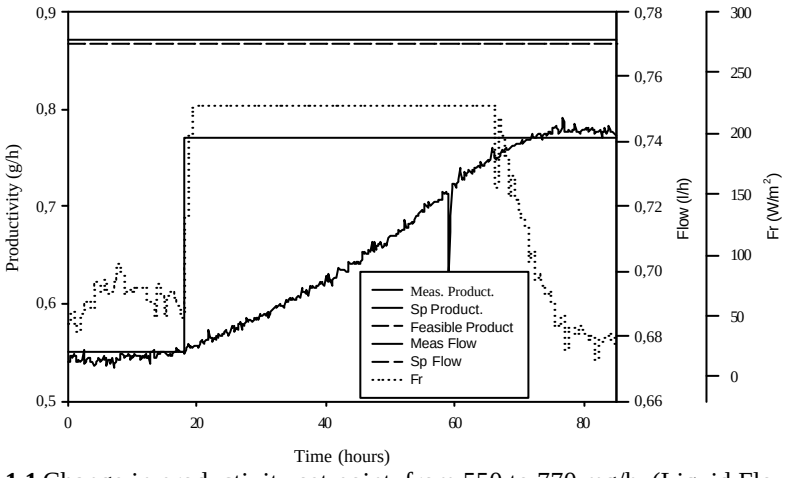


Figure 7.1.1.1 Change in productivity set-point, from 550 to 770 mg/h. (Liquid Flow 770 mL/h)

7.1.2 Decrease in the production Set point

Similarly as it was done in the case of an increase in the productivity set point, a decrease in the productivity se point was also tested. As soon as the set point is decreased, the control system decreases the light intensity (Fr), and as a result of the consequently decreased growth rate, the biomass is washed out from the bioreactor until the new biomass productivity set point is reached. Near that point the light intensity is increased so as to increase again the biomass growth rate. The light intensity is periodically increased-decreased so as to maintain the biomass productivity around the set point. The oscillations tend to stabilize with time as expected.

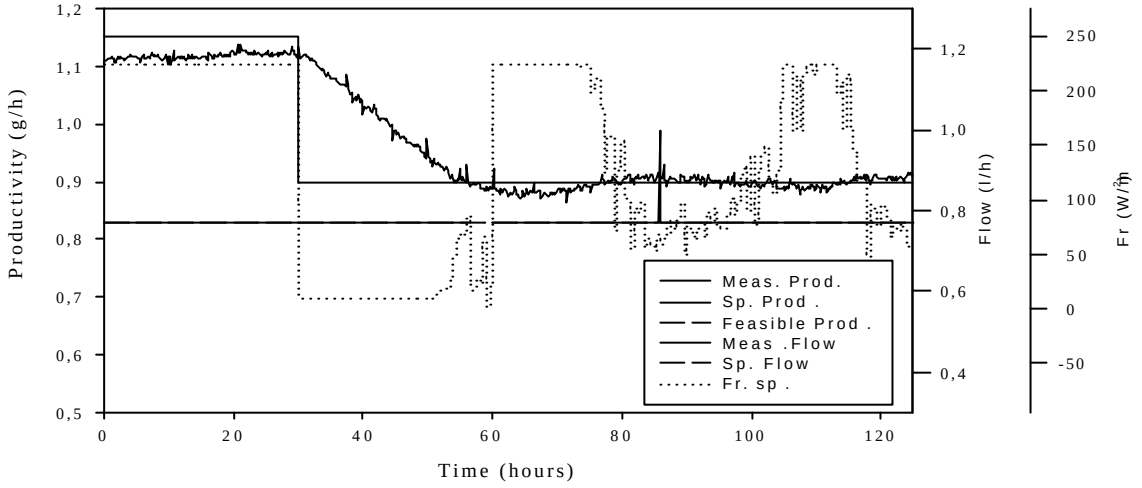


Figure 7.1.2.1 Change in productivity set-point, from 1200 to 770 mg/h. (Liquid Flow 770 mL/h)

7.2 Flow Control

Another of the actions that the user can perform is the modification of the liquid flow rate to the bioreactor. Increasing or decreasing the flow rate has a direct impact on the productivity of the bioreactor which the control system has to compensate. For example if the flow rate is decreased, the biomass concentration in the bioreactor has to increase so that the bioreactor biomass productivity is maintained.

Figure 7.2.1 shows that when the flow tae is deceased, at 20 h, the calculated productivity is immediately reduced. At this point the program recalculates what is the maximal productivity that can be reached at this flow rate, taking into account the biomass performance and the bioreactor characteristics through the knowledge model.

The calculations result in a feasible productivity set point, different from the user defined set point, but which is the maximum productivity that can be reached at this flow rate, in this bioreactor. This feasible productivity set point is higher than the productivity obtained at that flow rate and biomass concentration, and consequently it increases the light intensity in order to obtain a higher biomass concentration in the bioreactor and consequently an increased productivity. As in previous cases, as soon as this level of productivity is approached, the light intensity is decreased again, and is only slowly varied to maintain the biomass productivity at the newly reached set point. In summary in the tests performed, the control system has operated as expected. Therefore its implementation is considered successful. However it is considered that the stability of the reached set point can be increased by fine tuning the values of some of the control parameters. This operation will be done during the following tests to be done with the bioreactor operating interconnected to the III compartment.

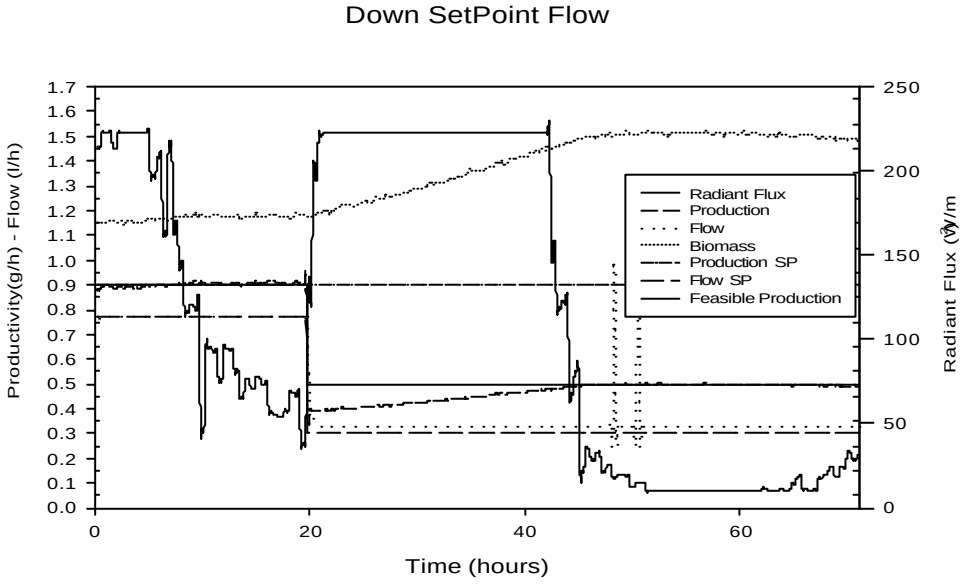


Figure 7.2.1 Modification of flow rate set-point, from 0.77 to 0.3 l/h.

8 FINAL SOURCE CODE

The final version of the software source code is listed in the following pages, for documentation purposes.

8.1 CtrlSpiru.c

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

        NAME            CTRLSPIR (EX: CONTROL.C)

        AUTHOR          BINOIS C    (modified by FULGET N. ADERSA)

        DESCRIPTION
                CONTROL PROGRAM listing file

        UPDATES
                20-09-95
                30-11-98  Modifications for the new reactor
                28-06-99  New Spirullina Control Routine
                17-09-99  UPGRADE OF THE NEW SPIRULINA CONTROL ROUTINE
*****************************************************************/

#include "ctrlspir.h"
#include "lspc.h"

#include <math.h>
#include "time.h"
 double ALPHAFILTERcxa = 0.3;
 double ALPHAFILTERnitrate = 0.3;
 double ALPHAFILTEREb = 0.3;
 double ALPHAFILTERpH = 0.3;
 FILE *fp;
 float spp,spf,cap,biof,adv,*pf ;
 int first_time_spiruline=1,pump_number;
 double init_spir;
 double cxaaverage; //new model for calculate cxa average
 int countav; // counter for do the cxa average
 int borrar;
 int counter_control;
 double trace[50];
 //fI (fraction illuminated)=VOLUME_LIGHT /VOLUME_TOTAL  (53/77
 #define fI 0.6883116
```

```
/*
       time_t  ltime;
       char    buffer[100];
       struct  tm  *dt;
       double  minute;
       //time(&ltime);
  */

       //      timt=localtime(&ltime);


/*-------------------------------------
       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_spirulina()
       {
       REACT   init_react;
       double  delta;
    int     jj;
```

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

        sprintf(cxa.name,"LOC5443");
        sprintf(nitrate.name,"LOC5459");
        sprintf(cal_nitrate.name,"DI--0125"); //OJO

        sprintf(Eb.name,"LOC5457");
        sprintf(Fr.name,"LOC5445");
        sprintf(temperature.name,"LOC5453");
        sprintf(pH.name,"LOC5455");
        sprintf(act_pompe.name,"LOC5447");
        sprintf(cal_pump.name,"LOC5449");
        sprintf(act_lamp.name,"LOC5461");

        sprintf(cons_prod_nom.name,"LOC5431");
        sprintf(cons_prod_real.name,"LOC5435");
        sprintf(qe_nom.name,"LOC5433");
        sprintf(qe_real.name,"LOC5437");
        sprintf(production.name,"LOC5439");
        sprintf(flow.name,"LOC5441");
        sprintf(prod_mod.name,"LOC5451");



        /*  Variables initialisation   */

        //only value for init
        biof=1.3;
        acq_vars_spirulina();

        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.value=init_react.Fr;
    fill_struct_var(&cxa);

/*
    *pf=Fr.value;
        wr_ascon("ASCON5.GPS",(&Fr),pf);
    cons_prod_real.sp=cons_prod_nom.value;
        *pf=cons_prod_real.sp;
        wr_ascon("ASCON5.GPS",&cons_prod_real,pf);
```

```
            qe_nom.sp=qe_nom.value;
               *pf=qe_real.value;
            wr_ascon("ASCON5.GPS",(&qe_real),pf);
         */

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

               init_spir=0;

        /* initialisation timer PFC */

                next_pfc_sp=1; //Do control when init the MCS


        /* counter to do average biomass */
                countav=0;   //init vars for calculate  biomass average
             cxaaverage=0;


                //MANUAL VARS INITIALISATIon
                      fp=fopen("SpirVal.txt","r");
                   if(fp==NULL)
                        {
                        display_error("Error     recovering     old     manual     values
(SpirVal.txt)");
                    display_error("Recovering standard manual values...");
                    spp=0.5;
                        spf=1.16;
                        biof=1.3;
                        cap=0.06;
                        cal_pump.value= cap;
                        adv=-0.068;

                        }
                       else {
                // Recover old manual values
                      fscanf(fp,"%f",&spp);
                      fscanf(fp,"%f",&spf);
                      fscanf(fp,"%f",&cap);
                      fscanf(fp,"%f",&adv);
                   fscanf(fp,"%f",&biof);
```

```
                cal_pump.value=cap;
                fclose(fp);
                first_time_spiruline = 0;
                }


// Init saved Setpoints
                cons_prod_nom.value=spp;
        qe_nom.value=spf;



           counter_control=0; //counter iteration control


        control_spiru();
        _clearscreen (_GWINDOW);
        display_status(" ");
        }


/*********************************************/
/*  Send vars for manual values send_vars_man */
/*********************************************/

void send_vars_man()

{ //Send modified parameters to control software
         // Actualization of SP Production
                *pf=cons_prod_nom.value ;
        wr_ascon("ASCON5.GPS",(&cons_prod_nom),pf);
        // Actualization of SP Flow
           *pf=qe_nom.value;
           wr_ascon("ASCON5.GPS",(&qe_nom),pf);
       display_status("");
}
```

```
/*-------------------------------------------------------
       variables acquisition for new controllers 26/11/98
-------------------------------------------------------*/

acq_vars_spirulina()
{
       REACT init_react;
       display_status("Acquisition of variables ...");


    // Read Biomass
       rd_ascon("ASCON5R.GPS","AI5001",pf);
       cxa.value=*pf;
       cxa.value= (cxa.value * biof); //Value for translate absorvance in biomass



        //Calculate biomass average
          cxaaverage=cxaaverage+cxa.value;
          countav++;
          if(countav<=0) cx=cxa.value;else cx=(cxaaverage/countav);


       //Calculate biomass sp for show (not used in control)
         if (qe_nom.value>0) cxa.sp=cons_prod_nom.value/qe_nom.value;

       //Measured Production // Two values monitorised for test
       production.sp=cx*qe_real.sp;
          production.value=cx*flow.value;


       // Read Temperature
       rd_ascon("ASCON5R.GPS","AI5003",pf);
          temperature.value=*pf;



   // Read PH
       /*rd_ascon("ASCON5R.GPS","AI6011",pf);
       pH.value=*pf;*/

    // Read Output Pompe
       rd_ascon("ASCON5R.GPS","AO5023",pf);
          act_pompe.value=*pf;
```

```
           // Read Nitrate
              if(!cal_nitrate.value)
                     { rd_ascon("ASCON5R.GPS","AI5009",pf);
                      nitrate.value=*pf;
                     }

              // Read Light //Only to know what is the real %action of controller
              rd_ascon("ASCON5R.GPS","AO5025",pf);
                 act_lamp.value=*pf/100;

                 /*Eliminate because if we would know the value of Fr in the controller
we change
                 the Fr.value used for control */
                 /*
                 act_lamp.value=*pf;
                 act_lamp.value=act_lamp.value/100;
              lightcal(&init_react,CAL_FR);
              Fr.value=init_react.Fr; */

       //Translate %action of controller in l/h
       // Read Measured Flow - Testing what pump is running //Modificate for to do by
Digital var test
                 rd_ascon("ASCON5R.GPS","AO5019",pf); //Read Input Pump I
                 flow.value=*pf;
              pump_number=1;

               if (flow.value<=0)
               { rd_ascon("ASCON5R.GPS","AO5021",pf);
                 flow.value=*pf;
                 pump_number=2;} //Read Input Pump II

               if (flow.value<=0)
               {display_error ("\nError, input pumps off!");
                flow.value=1; // To prevent problems with control
                return -1;
               }
                else {   flow.value=((flow.value)*(cal_pump.value))+adv;}//  Translation
controller action of pump(%) into flow (l/h)

          /*Send to the control software the manual values */
               send_vars_man();

       }
```

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

       void send_vars_spirulina()
           {

              display_status("Updating C4 vars ...");

              *pf=Fr.value; //Fr value
           wr_ascon("ASCON5.GPS",(&Fr),pf);

              *pf=cons_prod_real.sp; //Setpoint productivity
            wr_ascon("ASCON5.GPS",(&cons_prod_real),pf);

           *pf=qe_real.value; //Feasible Flow
           wr_ascon("ASCON5.GPS",(&qe_real),pf);

              *pf=production.sp; //Measured Production (biomass*Feasible flow)
           wr_ascon("ASCON5.GPS",(&production),pf);

              *pf=prod_mod.sp; //Model biomass production
           wr_ascon("ASCON5.GPS",(&prod_mod),pf);

              *pf=act_pompe.sp; //Output pump , sended but not used in controllers
                                          //because controllers has they own control
for output pump from input pump
            wr_ascon("ASCON5.GPS",(&act_pompe),pf);

           *pf=flow.value;  //Measured flow
           wr_ascon("ASCON5.GPS",&flow,pf);

           *pf=act_lamp.sp; //% Action light controllers
           wr_ascon("ASCON52.GPS",&act_lamp,pf);


              cal_pump.value=cap; //Values of calibration %Pump ---> l/h
              *pf=cal_pump.value;
              wr_ascon("ASCON5.GPS",&cal_pump,pf);


       /*
              *pf=pH.value;
           wr_ascon("ASCON5.GPS",(&pH),pf);
```

```
            *pf=Eb.value;
            wr_ascon("ASCON52.GPS",&Eb,pf); */


         display_status(" ");
           }


      /*-------------------------------------
            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_MAIN*SAMP_SPIRU));        /*        Abans
TSAMP_SPIRULINA */
            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_MAIN * SAMP_SPIRU));
       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
-------------------------------------*/

double average_var(VARS *diff_var, int diff_time)
    {
    int j;
    int i_samp,nb_samp;
    double average, aux1, aux2;

    aux1 = diff_time * (double) 60;
    aux2 = (double) TSAMP_MAIN * (double) SAMP_SPIRU;
    average=0;
    nb_samp=ceil(aux1/aux2);
    for(j=0;j<nb_samp;j++)
       {
       i_samp=(diff_var->i-j)&NB_SAMP;
       average+=diff_var->val[i_samp];
       }
    average/= (double) 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_MAIN * SAMP_SPIRU));
       for(j=0;j<nb_samp;j++)
              {
              i_samp=(diff_var->i-j)&NB_SAMP;
              average+=pow(diff_var->val[i_samp],2);
              }
    average/= (double) 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_MAIN * SAMP_SPIRU));
       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_MAIN * SAMP_SPIRU));
       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);
        }


/*************************************************************/
/*          NEW CONTROL PROCESS 28-6-1999                    */
/*                                                  UPGRADE   16-9-1999
*/
/*************************************************************/


int control_spiru(ScreenViews active_reactor)
        {
        FILE *fp;
        int retval = 0;
        char buffer[300], buffer2[300];
        REACT   react;

        double  dt, var_out[3];
```

```
 display_status("Reading C4 vars for run control...");
 acq_vars_spirulina();


 display_status("Control C4 running ...");

if(!(next_pfc_sp--))
    {

    //test iteration control
       counter_control++;

    /* mean biomass concentration */
    //      cx=average_var(&cxa ,10) ;  //Changed &cxa.value in &cxa
 //Problems with cxa average new average calculation
   countav=0;
      cxaaverage=0;
      if(cx<=0) cx=cxa.value; // cx : var wich is the average of biomass

    //Removed in last routine control
    /* illuminated surface fraction */
    /* fI = VOLUME_LIGHT / VOLUME_TOTAL; */

    /* control period expressed in h */
      dt = DT / 60.;

    /* light spirulina production control */
       lspc(cons_prod_nom.value, cx , qe_nom.value,
                &Fr.value, qe_real.value, &sm_sup, &cons_sup,
                VOLUME_TOTAL, fI, dt, LAMBDA, &init_spir, var_out,trace);


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

    /* real flow rate set point */
           qe_real.sp = var_out[1];

    /* pump setpoint sended to P100 controller */

           //Not used in the controllers
```

```
                act_pompe.sp=qe_real.sp/cal_pump.value;

        /* model output of spirulina growth */
                prod_mod.sp = var_out[2];

        /* feasible production setpoint */
                cons_prod_real.sp = var_out[0];

        /* Calculate production-------yo */
              production.sp=cx  * qe_real.sp;  // Measure production

                  //Control calculate qe_real.sp and need qe_real.value
                  qe_real.value=qe_real.sp; //Feasible flow value not read YO

                next_pfc_sp=DT;
                retval = 1;
                }

    /* Send ALL values to the controllers */
         send_vars_spirulina();
    send_vars_man();


     display_status("");



        return retval;
        }


/*************************************************************************/
/*                              CALIBRATION LIGHT                        */
/*************************************************************************/


#ifndef ADERSA
lightcal(REACT *react, int mode)
#else
lightcal(react, mode)
REACT *react;
int mode;
#endif
   {
        /* The following parameters come from ADERSA TN44.1 */
```

```
        double  a = 289.0;      double  b = 54.56;      double  c = -24.19;
        double  ymin = 5.;      double ymax = 310.;
        double  xmin = .2371;   double xmax = 1;

        double  x, y;

        switch(mode) {
               case CAL_FR:
               {
               /*  FR determination -------------------------*/
               x = max(xmin, min(xmax, act_lamp.value));
               react->Fr = a*x*x + b*x + c;
               break;
               }

               case CAL_ACT:
               {
               /*  light controller action determination ----*/
               y = max(ymin, min(ymax, react->Fr));
               act_lamp.sp = (-b + pow(b*b - 4*a*(c-y) , 0.5)) / 2. / a;
               break;
               }

               default:
               {

               display_error("Error in routine lightcal ...\n");
               display_error("*** Program terminated ***\n");

               exit(0);
               }
        }

    }



/*-------------------------------------------------------------
       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;   //Old Values for 7 litres Air Lift
         //    double  b_lamp = -33.03;
             double  Fr;
         switch(mode) {
               case CAL_FR:
               //{  //FR determination -------------------------
               //Fr = exp(a_lamp * act_lamp.value) + b_lamp; //Old Model 7 lt
               Fr   =   (350.93   *   (act_lamp.value*act_lamp.value)     )-   (23.858   *
act_lamp.value)- 2.256;
               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; // Old model for 7
lt
               //  act_lamp.sp=(act_lamp.sp * 100);
               if ((react->Fr) >225) act_lamp.sp=0.85;
               else{ if ((react->Fr) < 9.5) act_lamp.sp=0.25;
                    else{act_lamp.sp = (-0.000005 * (react->Fr) *(react->Fr)) + (0.0038
* (react->Fr)) + 0.2359;}
                   }

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

          }
```

```
*/


Melissa.h
/*********************************************************************

NAME            MELISSA.C (ex. SPIRULIN.C)

AUTHOR           BINOIS C      (modified by FULGET N. ADERSA & PEDRO PONS)

DESCRIPTION
        MAIN PROGRAM listing file

UPDATES
    20-09-95, 11-05-97

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

#ifndef ADERSA
#include <graph.h>
#include <process.h>
#endif
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <dos.h>
#include <direct.h>
#include <io.h>
#include <conio.h>
#include <bios.h>
#include <math.h>

#include "userdef.h"
#include "ctrlspir.h"
#include "ctrlnitr.h"
#include "help.h"
#include "melissa.h"
#include "melfct.h"
#include "screen.h"
#include "pargene.h"
#include "simfile.h"
int initkey=0;
#define _debugging_vars

extern double average_var (VARS*,int);
```

```
extern datagraph spdata, actdata, ntdata;
unsigned long checkfloppy (void);


#define __ALL_REACTORS
int     my_interrupt();
GPS_FILE *gps, *open_file_gps(), *activ_grp_gps();

int diskfull=0, diskwarning=0;  /* Controle the capacity of the disk */

int min_spir,       /* Count the minutes left to take a sample  */
    min_lique,      /* for each compartment                     */
    min_rhodo,
    min_nitri,
    min_glob;

int SAMP_SPIRU,      /* Define periods for taking samples */
    SAMP_RHODO,
    SAMP_LIQUE,
    SAMP_NITRI,
    SAMP_GLOB;

int pointer;

VARS    essai;
void alarms (void);
int control_process (ScreenViews active_reactor);
void keybdrv (ScreenViews *react, char chr);
void disk_storage (int input);
void parameters_adquisition(void);
void set_defaults(void);
void init_samples (void);
void check_disk (void);
void move_files( char*, char*);
void remove_file (char *name);

/*#define _testing_screens*/
#ifndef _testing_screens

main()
{
    char    chr;
    ScreenViews active_reactor=principal;  /* Indicates the info to show */
    FILE *fout_spirulina;
```

```
/*-------------------------------------
    screen and graphical initialisation
-------------------------------------*/
    InitializeGraphicalStructures ();

#ifndef ADERSA
       screen_init();
#endif

/* ====================================
       log file initalisation
  ==================================== */

    check_disk();

       init_log_file();

/*-------------------------------------
       set interruption
-------------------------------------*/

       if( signal(SIGINT,my_interrupt) == (int(*)())-1)
              {
              _outtext("\nCouldn't set SIGINT *** Program Terminated ***\n");
              exit(0);
              }


/*-------------------------------------
       timebase synchronisation
-------------------------------------*/

       wait_time(1);

/*-------------------------------------
       open groups and active one
-------------------------------------*/

    gps=open_file_gps();
       gps=activ_grp_gps();
    set_gps(gps);
/*-------------------------------------
       variables and output file initialisation
-------------------------------------*/
       parameters_adquisition();
```

```
               init_vars();
               init_alarm();
            result(&active_reactor,0);
            init_samples();

        /*-------------------------------------
               waiting loop
        -------------------------------------*/
               do
               {
               if (!timebase_main())           /* Timer principal */
                       {
                       int in;

                       check_disk();
        #ifndef _debugging_vars
                       check_network();             /* The check is going to
                                                                                be made
once a minute */
                       alarms();
        #endif
                       in=control_process(active_reactor);
                       disk_storage(in);
                       }
               if(kbhit())
                       {
                       chr=getch();
                       keybdrv(&active_reactor, chr);
                       }
               }
               while(1);
        }
        #else

        main()
        {

        screen_init();
        while (!kbhit());
        _setvideomode (-1);
        }

        #endif
        /*-------------------------------------
               interruption of programm
        -------------------------------------*/
```

```
int     my_interrupt()
        {
        char    ch;
        signal(SIGINT,SIG_IGN);
        _clearscreen (_GWINDOW);
        _outtext("Terminate processing ? ");
        ch=getch();
        if((ch=='y')||(ch=='Y'))
                {
          close_grp_gps();
                _outtext("\nbye.... *** Program Terminated by user ***\n");
         wait_time(5);
#ifndef ADERSA
                _setvideomode(_DEFAULTMODE);
#endif
                exit(0);
                }
        if( signal(SIGINT,my_interrupt) == (int(*)())-1)
                {
                _outtext("\nCouldn't set SIGINT *** Program Terminated ***\n");
                exit(0);
                }
        _outtext("Continue...\n");
        return;
        }

/* ================================
    Keyboard driver routine.
================================ */


void keybdrv (ScreenViews *react, char chr)
{
int modified=0;
int new=1;

if (!chr)   /* Tecles de funcio */
    {
    chr = getch();
    switch (*react)
    {
    case status:       /* F10 turn on/off alarms */
       {
           if (chr == 0x44) {activated_alarms = (activated_alarms ? 0:1);
```

```
                        ++modified;
                        }
            break;
        }
        case simulationParams :
                {
                if (chr== 0x3f) {
/* F5 */
                        InputConc(0);
                        ++modified;
                        }
                if (chr== 0x40) {
/* F6 */
                        InputConc(1);
                        ++modified;
                        }
                if (chr== 0x41) {
/* F7 */
                        InputStep (1);
                        ++modified;
                        }
                if (chr== 0x42) {
/* F8 */
                        InputStep (0);
                        ++modified;
                        }
                }
                break;

    case filemanagement:         /* File manageing submenu */
        {
        if (chr== 0x3b) { *react= help_1;           /* F1 */
                  ++modified;
                }
        if (chr== 0x3f) { move_disk();                /* F5 */
                         ++modified;
                }
        if (chr== 0x40) { move_files ("log.txt", "A:log.txt"); /* F6 */
                }
        if (chr== 0x41) { remove_file ("log.txt");      /* F7 */
                        }
        if (chr== 0x42) { directory ();      /* F8 */
                         *react = principal;
                        }
        if (chr== 0x43) { copy_disk();                /* F9 */
                         ++modified;
```

```
            }

    } break;
default:            /* Main menu */
    {
    if (chr== 0x3b) { *react= help_1;           /* F1 */
              ++modified;
            }
    if (chr== 0x3c) { *react= status_spir;           /* F2 */
              ++modified;
            }
    if (chr== 0x40) { *react= filemanagement;           /* F6 */
              ++modified;
            }
    if (chr== 0x3e) { *react = spirulina;       /* F4 */
          initkey=1;
              ++modified;
            }
    if (chr== 0x6c) { *react = gspirulina;       /* Alt+F5 */
                    ++modified;
                  }
    if (chr== 0x62) { *react = igspirulina;       /* Crt+F5 */
                    ++modified;
                  }
    if (chr== 0x3d) { *react = nitrifying;       /* F3 */
              ++modified;
            }
    if (chr== 0x6d) { *react = gnitrifying;       /* Alt+F6 */
                    ++modified;
                  }
    if (chr== 0x63) { *react = ignitrifying;       /* Crt+F6 */
                    ++modified;
                  }
    if (chr== 0x41) {
            //test *react = reactor_3;       /* F7 */
            next_pfc_sp=0;
            control_spiru();
            ++modified;
            }
    if (chr== 0x42) { *react = reactor_4;       /* F8 */
              ++modified;
            }
    if (chr== 0x43) {
            *react = simulation1;       /* F9 */
            ++modified;
```

```
                }
            if (chr== 0x70) {
                *react = simulation2;       /* Alt+F9 */
                ++modified;
                }
            if (chr== 0x71) {
                *react = simulationParams;      /* Alt+F10 */
                ++modified;
                }
            if (chr== 0x66) {
                *react = simulation3;       /* Ctr-F9 */
                ++modified;
                }
            if (chr== 0x67) {                           /* F11 */
                /* The execution of the simulation must be performed */
                double H;

                H = tsim / sim_dt;                /* Number of steps = hours /
sampling per. */
                                            /*  Here  we  have  to  call  to  the
simulation routines. */
                display_status ("Simuling ...");
                modspiru2 (H, FCr0, FFRH, FdilH, FCiH, FCrH, FmasbioH, FchonspH);
                PrintRes(H);
                display_status ("Simulation finished \a");
                display_status ("");
                ++modified;
                }
            if (chr== 0x68) {                             /* ALT+F1 */
                screen_init();
                ++modified;
                }

       //MARK
                if (chr== 0x54) {                                  /* SHIFT+F1 */

                    *react = inputsetpoint;

                ++modified;

                 }     /* Manual values for Spirullina */
                break;
```

```
}

}   /* End of switch */
    } /* End of if keypressed = F* */
    else
    {
    if (chr == 27) { *react = principal;
                    ++modified;
                  }
          else
          {
          switch (*react)
              {
              case gspirulina :
              case igspirulina:
              case gnitrifying:
              case ignitrifying:
              case simulation1:
              case simulation2:
              case simulation3:
                     {
                     switch (chr)
                         {
                         case '+' : AdvanceScale ();
                                    break;
                         case '-' : GoBackScale();
                                    break;
                         case '*' : IncrementScale ();
                                    new=0;
                                    ++modified;
                                    break;
                         case '/' : DecrementScale();
                                    new=0;
                                    ++modified;
                                    break;
                         default : break;
                         }
                     break;
                     }


              }
          }       /* End of else */
    }
```

```
if (modified) { if (!result(react,new)) {result(react, new); }}
}


/* ====================================
       Local Loop
==================================== */

int control_process (ScreenViews active_reactor)
{
int out=0;
char buffer[100];



if ((--min_glob)==0)
    {
           /* Should call to global control */
    min_glob = SAMP_GLOB;
    }

if((--min_lique)==0)
       {
       if (control_lique(active_reactor)) out = out | 0x02;
    min_lique = SAMP_LIQUE;
       }

if((--min_rhodo)==0)
    {
    if (control_rhodo(active_reactor)) out = out | 0x08;
    min_rhodo = SAMP_RHODO;
    }

if((--min_nitri)==0)
       {
       if (control_nitrogen(active_reactor)) out = out | 0x04;
    min_nitri = SAMP_NITRI;
       }
//mark
//if((--min_spir)==0)
 //   {
   // if (control_spiru(active_reactor)) out = out | 0x01;
   control_spiru(active_reactor);
    out=out | 0x01;
   min_spir = SAMP_SPIRU;//OK
   // }
```

```
_clearscreen(_GWINDOW);

if (out & 0x01 && active_reactor == gspirulina) result (&active_reactor,0);
if (active_reactor != gspirulina && active_reactor != igspirulina &&
    active_reactor != simulation1 && active_reactor != simulation2 &&
    active_reactor != simulation3 )
    result (&active_reactor,0);
    #ifndef __Monitoring
    else
           next_control ();
    #endif

return out;
}


void alarms (void)
{
char buffer[100];

alarm_spiru();
alarm_lique();
alarm_nitri();
alarm_rhodo();

sprintf (buffer, "Accessing reactor alarms ...\n");
_outtext (buffer);
wait_time (1);

_clearscreen(_GWINDOW);
}


void disk_storage (int input)
{
FILE *fp;
double v1, v2;
char buffer[400], file[40];
time_t ltime;
struct tm *dt;

if (diskfull) return;

if (input & 0x01)       /* Spirulina output */ //write disk spirulina
```

```
            {
          strcpy (file,"sp");
          time(&ltime);
          dt=localtime(&ltime);
          dt->tm_mon++;
          sprintf(file,"./gpsdat/sp00%02d%02d.out",dt->tm_mon,dt->tm_mday);

            fp = fopen (file, "rt");
            fp = fopen ( )
          if (fp==NULL)
            { /* If can't open file -> new file */

                    char buffer[400];
                    fp = fopen (file, "at");  /* write heading */
                    //sprintf (buffer, "cxa_moy, nit_moy, production.sp, react.temp,
cons_prod_real.sp,  qe_real.sp, qe_real.value, Fr.sp, Eb.sp\n");
                    sprintf (buffer, "       FProd   FFlow   MProd   MFlow    FRSP ABSV
SMSUP  CONSUP SPProd SPFlow\n");
                    fwrite(buffer, 1, strlen(buffer), fp);

            }
            else    /* .... if could open, go to last position */
            {
                   fclose (fp);
                   fp = fopen (file, "at");
            }

          if (fp==NULL) { display_error ("Unable to open output file");
                        return; }
          //v1 = average_var(&cxa, 10);
          //v2 = average_var(&nitrate, 10);

            wait_time(2);
           sprintf (buffer, "%02d:%02d %6.4f %6.4f %6.4f %6.4f %5.1f %6.4f %6.4f %6.4f
%6.4f %6.4f\n",dt->tm_hour,dt->tm_min,
            cons_prod_real.sp,qe_real.sp,production.sp,flow.value,Fr.sp,cx,sm_sup,con
s_sup,cons_prod_nom.value,qe_nom.value);
            fwrite (buffer, 1, strlen(buffer), fp);
            fclose (fp);
            }

      if (input & 0x04)       /* Nitrifying output */
            {
          strcpy (file,"sp");
          time(&ltime);
          dt=localtime(&ltime);
```

```
            dt->tm_mon++;
            sprintf(file,"./gpsdat/nt00%02d%02d.out",dt->tm_mon,dt->tm_mday);

            fp = fopen (file, "rt");
            if (fp==NULL) { /* if can't open -> new file */
                            char buffer[400];

                            fp = fopen (file, "at");  /* write heading */
                            sprintf   (buffer,   "DO.value,   pH.value,   Temperature.value,
Temperature.sp    ,    Pressure.value,         AmmoniumConc.value,    NitrateConc.value,
AmmoniumConc.sp, NitrateConc.sp\n");
                            fwrite(buffer, 1, strlen(buffer), fp);
        display_status(" write heading");
                           }
                else    /* .... if could open, I go to last position */
                    {
                    fclose (fp);
                    fp = fopen (file, "at");
                    }

            if (fp==NULL) { display_error ("Unable to open output file");
                            return; }
            v1 = average_var(&cxa, 10);
            v2 = average_var(&nitrate, 10);
             sprintf (buffer, "%02d:%02d %f %f %f %f %f %f %f %f %f\n",dt->tm_hour,dt-
>tm_min,
               NitrDo.value , NitrpH.value , NitrTemp.value, NitrTemp.sp,
               NitrPressure.value,    NitrAmCon.value,  NitrNitrCon.value,  NitrAmCon.sp,
NitrNitrCon.sp);
               fwrite (buffer, 1, strlen(buffer), fp);
               fclose (fp);
            }

        }



        void parameters_adquisition(void)
        {
        FILE *fp;
        char input[100];
        int ok=0;
        float inputval=0;
        long inputlong;

        set_defaults();
```

```
fp=fopen(CONFIG_FILE, "rt");
if (fp==NULL) { /* Set defaults values */
               tech_log_file (" Warning ... no configuration file.");
               return;
               }

while (!feof (fp))
       {
       ok=0;
       fscanf (fp, "%99s", input);
       if (strcmp(input, ""))
               {
               char *uperinput;
               uperinput = strupr(input);
               if (!strcmp (uperinput, "FILTERCXA")) {
                       fscanf (fp, "%f", &inputval);
                       ALPHAFILTERcxa = inputval;
                       ok++;
                       }
               if (!strcmp (uperinput, "FILTERNITRATE")) {
                       fscanf (fp, "%f", &inputval);
                       ALPHAFILTERnitrate = inputval;
                       ok++;
                       }
               if (!strcmp (uperinput, "FILTEREB")) {
                       fscanf (fp, "%f", &inputval);
                       ALPHAFILTEREb = inputval;
                       ok++;
                       }
               if (!strcmp (uperinput, "FILTERPH")) {
                       fscanf (fp, "%f", &inputval);
                       ALPHAFILTERpH = inputval;
                       ok++;
                       }
               if (!strcmp (uperinput, "SAMPSPIRU")) {
                    fscanf (fp, "%i", &inputlong);
                    SAMP_SPIRU = inputlong;
                    ok++;
                    }
               if (!strcmp (uperinput, "SAMPRHODO")) {
                    fscanf (fp, "%i", &inputlong);
                    SAMP_RHODO= inputlong;
                    ok++;
                    }
               if (!strcmp (uperinput, "SAMPLIQUE")) {
```

```
                    fscanf (fp, "%i", &inputlong);
                    SAMP_LIQUE = inputlong;
                    ok++;
                    }
                if (!strcmp (uperinput, "SAMPNITRI")) {
                    fscanf (fp, "%i", &inputlong);
                    SAMP_NITRI = inputlong;
                    ok++;
                    }
                if (!strcmp (uperinput, "SAMPGLOB")) {
                    fscanf (fp, "%i", &inputlong);
                    SAMP_GLOB = inputlong;
                    ok++;
                    }
        if (!ok) { /* Error reading parameter file, keyword not valid */
                                    /* Set default values */
                    char buffer[200];
                    set_defaults();
                    /* Error message */
                    sprintf (buffer, "Unknown parameter : %s",input);
                    display_error(buffer);
                    _clearscreen(_GWINDOW);
                    }
                }
        }

}

/*  ==========================
        Set all the varibles that can be parametrized
        to a default value.
========================== */


void set_defaults(void)
{
ALPHAFILTERcxa = 1;
ALPHAFILTERnitrate = 1;
ALPHAFILTEREb = 1;
ALPHAFILTERpH = 1;
SAMP_SPIRU = 1;
SAMP_LIQUE = 1;
SAMP_RHODO = 1;
SAMP_NITRI = 1;
SAMP_GLOB = 1;
```

```
        }

        void init_samples (void)
        {
        min_spir = SAMP_SPIRU;
        min_lique = SAMP_LIQUE;
        min_rhodo = SAMP_RHODO;
        min_nitri = SAMP_NITRI;
        min_glob = SAMP_GLOB;
        }

        void check_disk (void)
        {
        int disk;
        struct diskfree_t df;
        unsigned long lliure;
        char buffer[100];

        display_status ("Checking disk ... ");

        _dos_getdiskfree(0, &df);   /* Ud per defecte */
        lliure    =    (unsigned    long)    df.avail_clusters    *    (unsigned    long)
df.sectors_per_cluster * (unsigned long) df.bytes_per_sector;

        if (lliure < 100000) {diskfull=1; }


        if (lliure < 400000) {
                            /* Warning */
                            display_error ("Disk nearly full !!!");
                            diskwarning = 1;
                            }
                            else
                            { diskwarning = 0;
                              diskfull = 0;
                            }
        if (lliure < 100000) {
                           /* Disable disk saving */
                           sprintf (buffer,"Disabling disk saving ...");
                           display_error (buffer);
                           return;
                           }
        return;
        }
```

```
unsigned long checkfloppy (void)
{
struct diskinfo_t diskinfo;
struct diskfree_t diskfree;
int st;
unsigned long lliure;

diskinfo.drive = 0; /* ud A: */
diskinfo.head = 0;
diskinfo.track = 1;
diskinfo.sector = 2;
diskinfo.nsectors = 1;

st = _bios_disk (_DISK_VERIFY, &diskinfo);
if (0x8000 & st) return -1;

_dos_getdiskfree(1, &diskfree);

lliure = (unsigned long) diskfree.avail_clusters *
        (unsigned long)  diskfree.sectors_per_cluster *
        (unsigned long) diskfree.bytes_per_sector;


return lliure;
}

int copia (char *or, char *des, unsigned long lliure)
{
FILE *o, *d;
char c;

o = fopen (or,"rb");
if (o==NULL)  {
              display_status ("Error : source file does not exist.");
              printf("\a\a");
              wait_time(2);
              display_status (" ");
              return -1;
              }
fseek(o, 0L, SEEK_END);
if (lliure <= ftell(o)) {
                      display_status ("Not enough space in disk :");
                      printf ("\a\a");
                      wait_time(2);
                      display_status (" ");
```

```
                        return -5;
                        }
        fseek (o,0L,SEEK_SET);

        d = fopen (des, "wb");
        if (d==NULL)  {
                     display_status ("Error opening destination file.");
                     printf("\a\a");
                     wait_time(2);
                     display_status (" ");
                     return -2;
                     }

        fread(&c, 1,sizeof(char), o);
        do  {
            fwrite(&c, 1, sizeof(char), d);
            fread (&c, 1, sizeof(char), o);
            } while (!feof(o) && !ferror(o) && !ferror(d) );

        if (ferror(o)) {
                       display_status ("Error during the copy process in source file.");
                       printf("\a\a");
                       wait_time(2);
                       display_status (" ");
                       return -3;
                       }
        if (ferror(d)) {
                      display_status  ("Error  during  the  copy  process  in  destination
file.");
                       printf("\a\a");
                       wait_time(2);
                       display_status (" ");
                       return -4;
                       }
        fclose (d);
        fclose (o);
        return 0;
        }


        void move_files (char *name, char *destination)
        {
        unsigned long lliure;

        /* Verify the floppy drive and free space */
        if (strcmp (destination, "NUL") && strcmp(destination, "nul") )
```

```
    {
    lliure = checkfloppy();
    if (lliure == -1) {
                    display_status ("Drive A: is not ready.");
                    printf ("\a\a");
                    wait_time(2);
                    display_status (" ");
                    return;
                    }
    }

if (copia(name, destination, lliure)==0)
       {   /* If ok => remove the original */
       if (remove(name)==-1)
              {
              sprintf (buffer,"Could not delete the file %s",name);
              display_status(buffer);
              printf("\a\a");
              display_status (" ");
              wait_time(2);
              }

       }

}

void remove_file (char *name)
{
char c;

display_status ("Are you sure ? (Y/N)");
printf ("\a\a");

while (!kbhit());
c = getch();
if (c!= 'y' && c!= 'Y')
              {
              display_status(" ");
              return;
              }


if (remove(name)==-1)
       {
       sprintf (buffer,"Could not delete the file %s",name);
```

```
                display_status(buffer);
                printf("\a\a");
                display_status (" ");
                wait_time(2);
                }
        display_status(" ");
        }

        /* Due to the DOS version, no more files can be passed as parameters to the
linker,
        so here will be included the spirulina simulator.
        */
        /*#include "modspiru.c"*/
        /* This file should be included here, but due to the size of the present file
        it will be included in help.c
        */
```