Simultaneous engineering project

GrowBotHub : Scheduling

Querro Thomas

Montuori François

Supervisor: Colin Jones

Contents

1 Introduction

1.1 Growbothub project

GrowBotHub is a multidisciplinary project which the goal is to grow vegetables in extreme environments. It is part of a bigger project named "IGLUNA: An habitat in Ice" coordinated by the European Space agency and gathering more than 120 students in 13 European countries. The goal of this mission is to build a viable habitat on the moon, and each country is in charge of an aspect of the habitat. Switzerland works on the nutritional aspect. The system diagram shown bellow is a good introduction to the multidisciplinary character of this project.

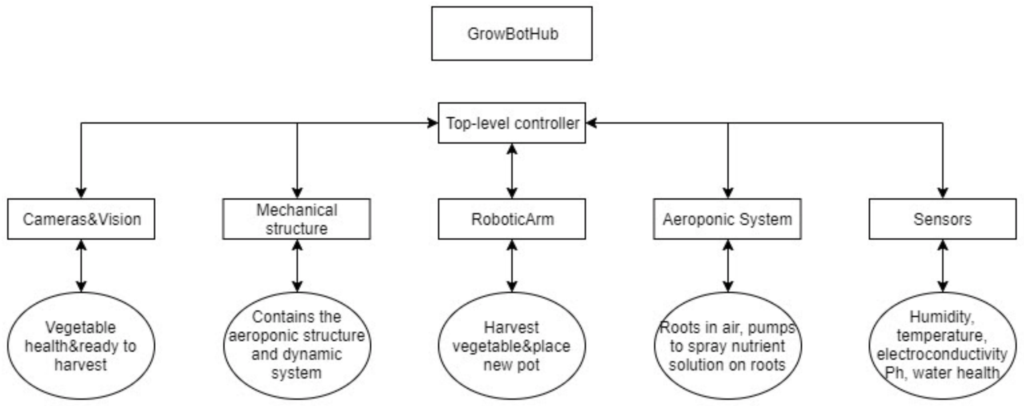

GrowBotHub
Top-level controller
Cameras&Vision
Mechanical structure
RoboticArm
Aeroponic System
Sensors
Vegetable health&ready to harvest
Contains the aeroponic structure and dynamic system
Harvest vegetable&place new pot
Roots in air, pumps to spray nutrient solution on roots
Humidity, temperature, electroconductivity Ph, water health

Figure 1: Aeroponics shelves system

1.2 Aeroponics system

To achieve this goal, GrowBotHub students imagined a mechanical structure on which various vegetables grow in aeropony. An automatic arm is in charge of moving the plants. Each Shelf contains a solution with a different concentration of nutriments. Along it's growth a vegetable needs different levels of nutriments, explaining why a plant would have to be moved several times from a shelf to another.

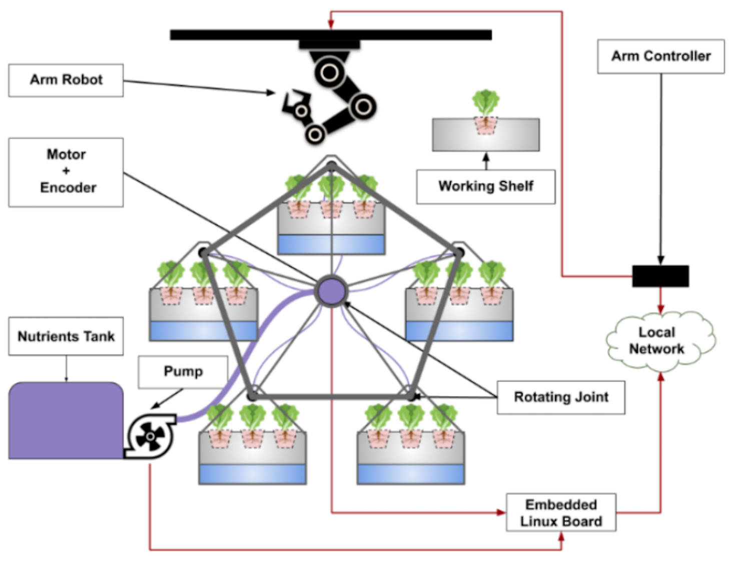


Figure 2: Aeroponic shelves system

A pump is in charge of directing nutriments from the main tank to the different shelves, and then water determines the final concentration in a shelf reservoir. The system has a carousel construction, allowing the shelves to be always horizontal even when moving. The arm robot can only work on the upper shelf. Each one contains 8 spots (i.e a maximum of 8 plants). The robot arm receives orders from the arm controller, which we plan to link to the scheduling code. When conceiving the mechanical system, the round-angle shade of the lunar modulus had to be taken into account, explaining the choice of a circular motion.

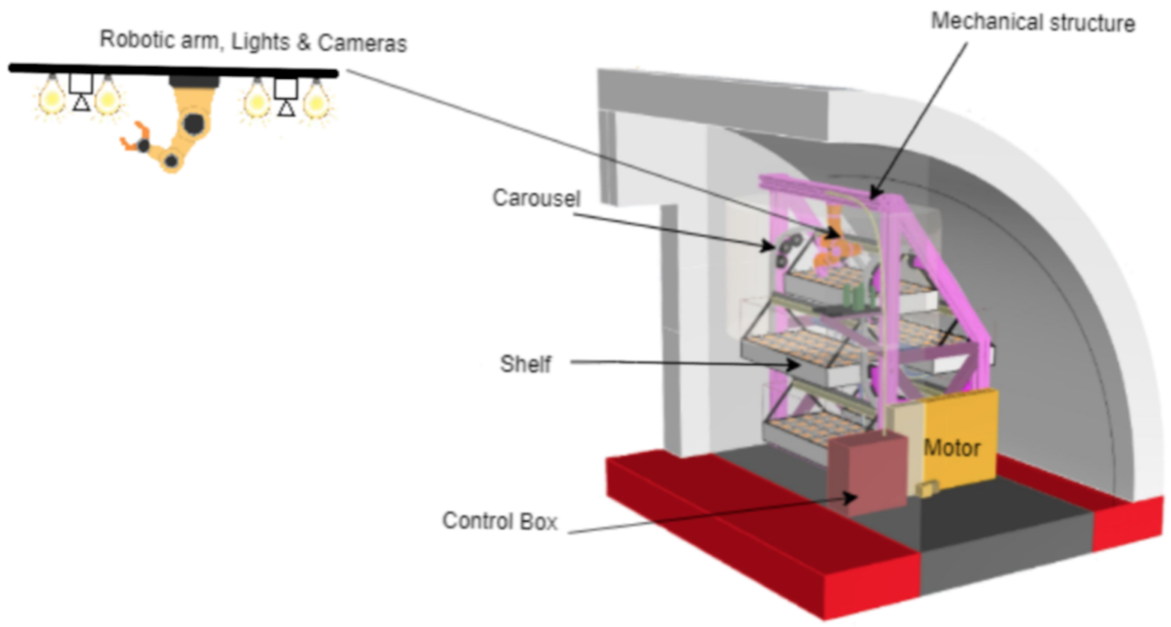


Figure 3: system overview

1.3 Prehexisting work

No such optimisations has be done to the system before. Before this scheduling was done, the vegetables were chosen in terms of preference and simplicity and the plants entered the system without a preference order or timing.

2 Goal and restrictions

In order to optimize the production of vegetables in the lunar modulus, we want to schedule the growth of the plants. This schedule will be then followed by the robot arm. Optimizing the vegetable's production implies two main factors: calories and time.

First, we want the system to produce a sufficient amount of nutriments for the astronauts. Ideally, the goal is to produce 30% of daily nutriments for 6 people. Nevertheless, this goal seems impossible with one modulus. This schedule might help to determine how many modulus we need to guarantee the number of daily calories aimed.

Secondly, we want the vegetables to be consumable the more rapidly as possible. The laps of time considered is between the first step where the grows are put into the system, and the final state where the vegetables/fruits are ready for consumption.

Eventually, there is a third parameter that we may take into consideration for the scheduling: variety. The idea behind this will of growing vegetables on moon is to assure good and fresh food for the astronauts, who eat a lot of dehydrated food. It is legitimate to think that vegetables such as potatoes would give a lot of calories, but growing only potatoes would definitely go against the idea of fresh vegetables to break the monotony of dehydrated food. For this reason variety is important to consider and will have to be quantified mathematically at some point.

Furthermore, we know that even if growth in aeroponics should be quite stable compared to a classic way to grow vegetables , we can not predict the exact going of a plant life cycle. We can use the some data to rely on, but we do know that the schedule is theoretic, thus, things may differ a little bit in real.

Finally, it would be a good thing that users could get the process progress in real time.

We do have to consider physical and natural restrictions. Even if a lot of fruits and vegetables are suitable to a growth in aeropony, growing vegetables in space implies spacial constraints and thus, we have to be sure that the size of the plants do not affect the carousel's movement.

3 Optimisation model

3.1 Choice of model

The system we want to optimize has to respect the shelves order, furthermore we have a production goal. We also want a planning reporting the movements from a spot to another between the shelves.

The structure of the problem lead us to what we call "factory scheduling". Vegetables can indeed be seen as products to "manufacture" and they go through different stages of production which are the shelves. Moreover, factory scheduling gives a planning production, here being the time (in hours) where the plants move between the shelves.

Hence, we decided to rely on this model, often used to model problems such as ours.

3.2 Problem modeling

To model the problem we first have to fix at 50 the number of plants per cycle, which allows to suppress the variable "number of plants" into a cycle. The plant type being specified before the program launch, we can thus optimise on the variables "time" and "calories". we decided to use them both in the subjective function, using the most accurate approximation of the ration calories over time. Practically, we can not use this parameter because we use a system of linear optimisation, thus the division of the parameters would make the problem non-linear.

We also could have fixed a number of calories that we would want to reach, and take as the only optimised variable time. Or else fix a time for a cycle (period) and optimise the amount of calories produced in this laps of time.

Nevertheless, we chose the first method because one important factor about our optimisation is to keep a quite important flexibility as regard to the plants to choose. Particularly in a perspective of diversification of the parameters and potentially adding more complicated ones such as vitamins, fibres, etc...). This is directly connected to the aspect of variety explained in §2.

We thus write the modeling of this problem as a linear optimisation problem, making the following assertions and constraints:

– Variables for the optimisation: calories and time of growth.

- Objective function = $max(a \cdot calories - b \cdot time)$
  The variables a and b will be detailed in the next section.
- The constraint of order of each shelf required by each type of plants.
- The constraint of maximum plants per shelf: a maximum of 8 plants simultaneously on each shelf.
- The growth constrain that obliges the plant to complete it's growth before counting the calories produced.
- Variety constraints entered by the user (for instance a minimum of one plant of strawberries among the 50). Even if such a choice implies a lower calories production.
- The constraint of 50 plants in the system for one cycle.
- The constraints of enter and exist of a plant from a shelf accordingly to it's type.

3.3 Study of the objective function

Ideally, we would have chosen the following objective function: :

$$max(\frac{calories}{time})$$

Nevertheless as explained in §3.2 we can not use it as it does not respect the linearity we need to use the solver. Thus we will try to approach it by a linear objective function:

$$max(a \cdot calories - b \cdot time)$$

a and b depend from the type of plants selected by the user.

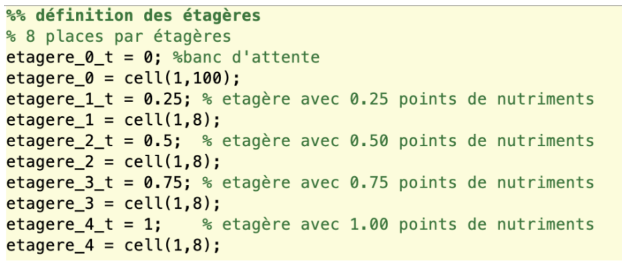
To determine a and b we model the function $\frac{x}{y}$ and approach it as $a \cdot calories - b \cdot temps$

4 Programming

To solve the problem explained we use processing tools. Moreover, due to the Covid-19 situation we had to do a simulation of the system via Matlab as the laboratories where inaccessible.

4.1 Preliminary work

The Matlab simulation was simulating the movements of the plants. A loop system was able to place them from a shelf to another accordingly their state of growth. To follow the state of growth of the plants, we imagined the notion of "points of nutriments". It allows to follow the growth of the plant after passing through the shelves. "0 represents a plant that did not begin yet his cycle of growth in the system, and 1 means the plants is ready to being consumed. Each 0.25 points of nutriments, the plant is ready to be moved from a shelf to another.



```
%% définition des étagères
% 8 places par étagères
etagere_0_t = 0; %banc d'attente
etagere_0 = cell(1,100);
etagere_1_t = 0.25; % etagère avec 0.25 points de nutriments
etagere_1 = cell(1,8);
etagere_2_t = 0.5;  % etagère avec 0.50 points de nutriments
etagere_2 = cell(1,8);
etagere_3_t = 0.75; % etagère avec 0.75 points de nutriments
etagere_3 = cell(1,8);
etagere_4_t = 1;    % etagère avec 1.00 points de nutriments
etagere_4 = cell(1,8);
```

Figure 4: extract from the simulation Matlab script: "points of nutriment"

We then used Gurobi optimizer the type of plants chosen and the entry time or exit time of the plants in each shelf. Due to the complexity of setting up the constraints of order, the program had to be translated into Python with a Google package named "OR-Tools"

4.2 Python and optimisation

We first define the number of shelves, the number of spots in each and the the number of plants of a cycle. Then we define the plants *type*. It is composed by six information given by the user: name, calories, number of hours in the first shelf, number of hours in the second, number of hours in the third, and finally number of hours in the fourth one. If a type of plant does not go through a certain stage (corresponding to a shelf), it take the value 0 for this stage. Nevertheless, a spot has to be free to move the plant directly to the next one.

We define the plants with the same information described in the plants *type*, with the difference that the values given from the third to the sixth column of the vector are not anymore the time passed in each stage but the hour of entry and exit. Zero being taken at the moment a plant enters the system. We also ad the following assertions:

- The plant has to stay in each shelf at least the minimal time defined by it's type. Nevertheless, it can stay longer if no spot is available at the moment.
- The plant must move from a shelf to another in the right order.
- If the plant can move from a stage to the next one, a spot has to be free on the next shelf before it moves.

Then we define the shelves, where the *Addcumulative* command that bothered us on Gurobi allows now to create an object with a capacity and a transfer by order and needs. Thus if a shelf has a free spot it "asks" a plant ready to come from the precedent stage. Finally, we give the users constraints: minimal number of plants from each type the system must contain within a cycle.

In a second step, we define the variables total_grow time and total_calories which are respectively the time taken by the 50 plants to finish their growth and the number of calories collected when all the plants finish their growth. With the completeness of these values the scheduling can be operated.

After obtaining the solution we use the data from the solver to create two text files containing:

- optimised variables: total_grow time & total_calories.
- Time schedule (hours) containing entries in each shelves. This data is given by 50 x 5 matrix: A row for each plant and in the columns the four hours of entry/exit plus the hour of harvest.

Also two Matlab documents to summarise the solvers data for the user.

4.3 Labiew and user interface

We use Labview for it ability to modify and create new codes according to their needs. Furthermore, linking Matlab functions via python is an major issue. Thus we also decide to choose Labview for its facility to link Matlab to Python.

We set up a user panel (see Annex). It allows to define the type of plants in order to put them in the system. Also the users constraints to define the minimal number of plants for each type. Then, the Matlab file in charge of finding the optimal objective linear function $max(\frac{calories}{temps})$, in function of the users choices that will give the values of a and b.

With that a new python code is written and launched, then the Matlab simulation takes up the slack. In the debug tab the user can find the different scripts, the Python solvers data and the approximation of the objective function. Also, in the "path" debug the user can choose where to download the different documents.

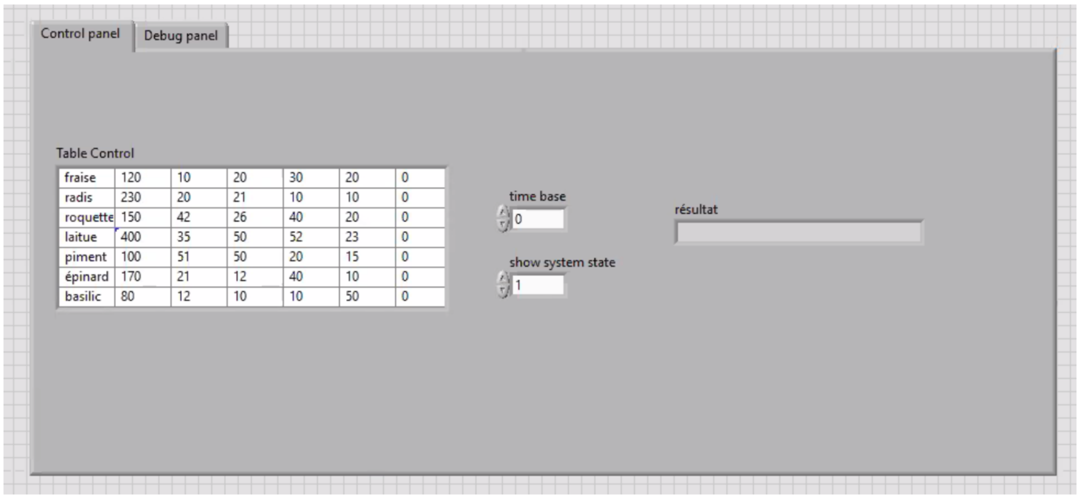


Table Control

fraise	120	10	20	30	20	0
radis	230	20	21	10	10	0
roquette	150	42	26	40	20	0
laitue	400	35	50	52	23	0
piment	100	51	50	20	15	0
épinard	170	21	12	40	10	0
basilic	80	12	10	10	50	0

time base: 0

show system state: 1

résultat

Figure 5: Labview - user interface

4.4 Matlab and simulate environment

We use Matlab for two distinct parts: The first one, presented in §4.2 finds a value for a and b according to the calories and times of growth.

$(a \cdot calories - b \cdot time)$ that approximates at best $(\frac{calories}{time})$

The second one is the simulation of the system. As we do not have the physical system and due to the fact the scheduling doesn't give the spot where the plant is at a time t, we simulate the moves and the growth via the file containing the entry/exist hours. Furthermore, the schedule does not need to know where the plant is on the shelves to give the entries hours between shelves. Then we can the the virtual system running and we obtain the spot for each plant that the robotic arm will need to know where to take and put the plants in the shelves.

4.5 Link to micro controller

After Matlab simulate the system he will send a matrix on a text file where each line is a plant and the column goes by pair : the hour when the plant enter the trail and where it will be placed in the trails. Thus with this the program into the micro controller will know which order give to the robotic arm. But our program doesn't take into account the delay of grow we can happen to a plant, so another checking system have been implemented to check with a camera the stage of grow of each plant and put a delay in the matrix we give if the plant take more time to grow.

5 Results and configurations

Together with the life science students involved in the project, we selected a list of seven fruits and vegetables to potentially implement in the system. These were chosen accordingly to their ability to grow in an aeroponics system, their nutritional properties and their fast time of growth. All this guarantying a certain variety in the type of fruits and vegetables.

We propose threw strategies or configurations of the problem leading to threw scheduling. The results presented must be taken qualitatively because they depend on the data used for each plant. Due to the Covid-19 situation, the data is estimated from researches on the internet, therefore they are not the result from our observations or measurements. Even though, we let the user the possibility to the user to adjust them from the control panel if necessary.

5.1 maximum calories per time configuration

The first approach of the scheduling let the program the maximum freedom to optimise the scheduling. Therefore the user does not put any constraints on the minimal number of each plant he or she wants. Practically, this is translated by a zero in the last column of the data for each plant. Here the program will try to get the maximum number of calories in the minimum time possible, without taking into account any parameter related to variety. This scheduling gives a final production of 11500kcal in a time of 7 days and 19 hours (187hours).

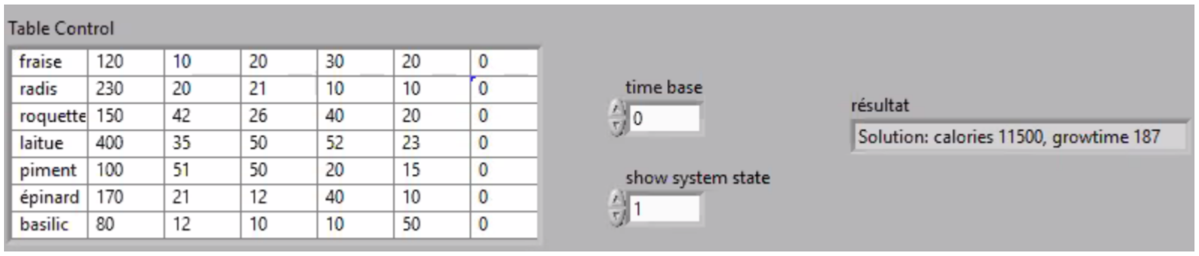


Table Control

fraise	120	10	20	30	20	0
radis	230	20	21	10	10	0
roquette	150	42	26	40	20	0
laitue	400	35	50	52	23	0
piment	100	51	50	20	15	0
épinard	170	21	12	40	10	0
basilic	80	12	10	10	50	0

time base: 0

show system state: 1

résultat: Solution: calories 11500, growtime 187

Figure 6: First configuration - user panel and main results

Looking a the figure 7, we can see that the result was obtain by growing 50 plants of type 1, corresponding to radish. We can conclude radish is the plant with the most efficient calories production among our list.

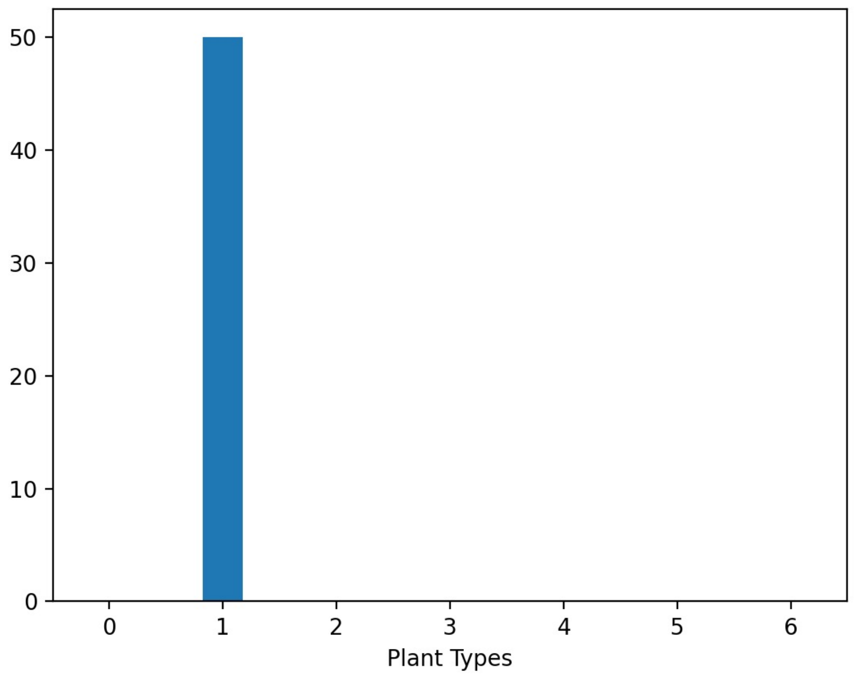


Figure 7: First configuration - type of plants distribution

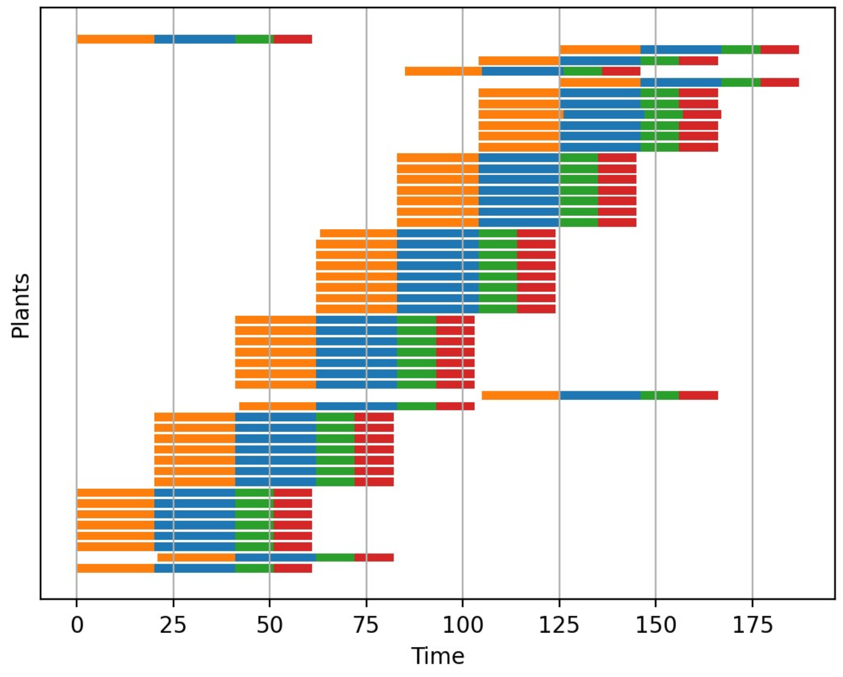


Figure 8: First configuration - schedule representation

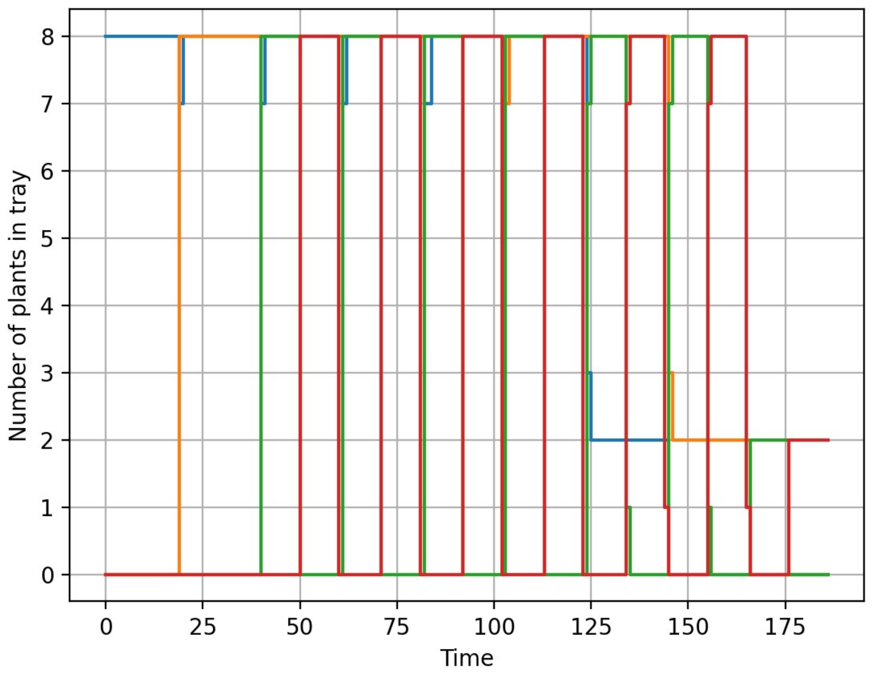


Figure 9: First configuration - schedule representation bis

Figure 8 & 9 present the detailed scheduling of the optimisation. The color code to read the graphs is the following: first stage: orange; second stage: blue; third stage: green; fourth stage: red

5.2 Mixed configuration

We call the second configuration the "mixed" one: we choose about half the repatriation of the plants, meaning we want at least 4 plants of each, and the rest is left up to the program. Hence we constrain a minimum of 4 plants of each. We can see in fig. 11 that the "free part" is filled with radish again. We can see that the minimum of 4 plants of each type is respected. The result given is a total of 10060kcal in 11 days and 20 hours (284 hours).

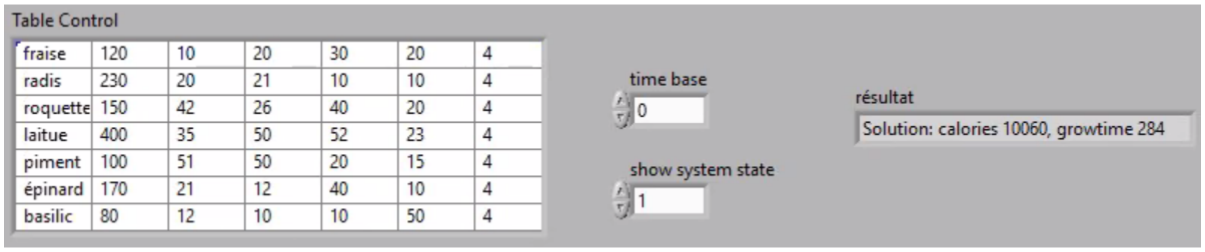


Table Control

fraise	120	10	20	30	20	4
radis	230	20	21	10	10	4
roquette	150	42	26	40	20	4
laitue	400	35	50	52	23	4
piment	100	51	50	20	15	4
épinard	170	21	12	40	10	4
basilic	80	12	10	10	50	4

time base: 0

show system state: 1

résultat: Solution: calories 10060, growtime 284

Figure 10: Second configuration - user panel and main results

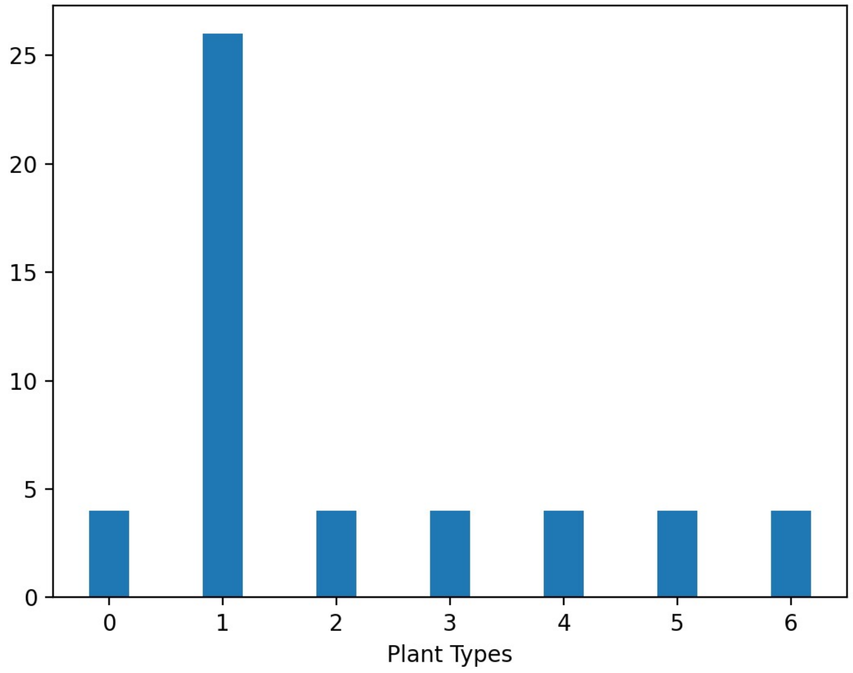


Figure 11: Second configuration - user panel and main results

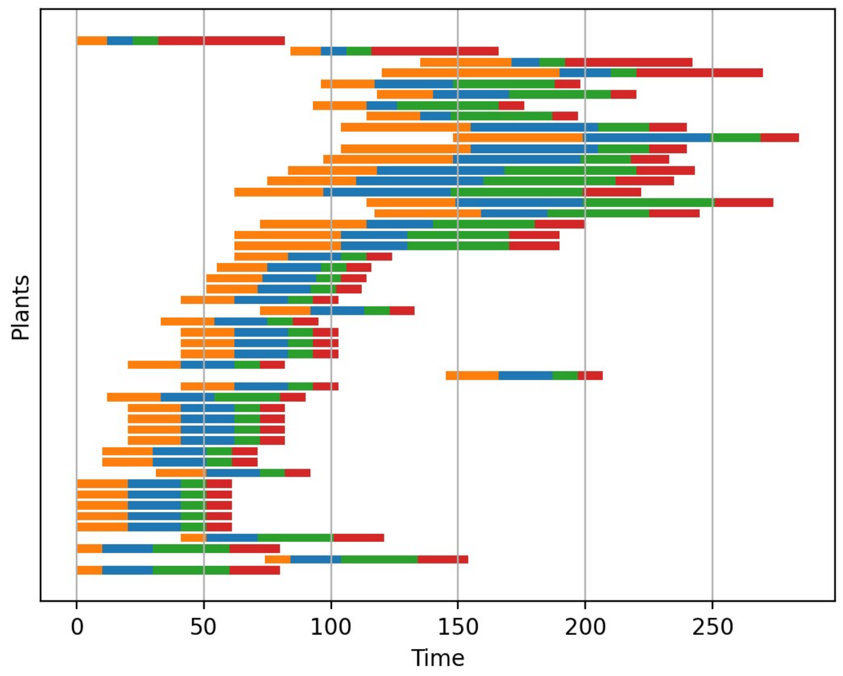


Figure 12: Second configuration - user panel and main results

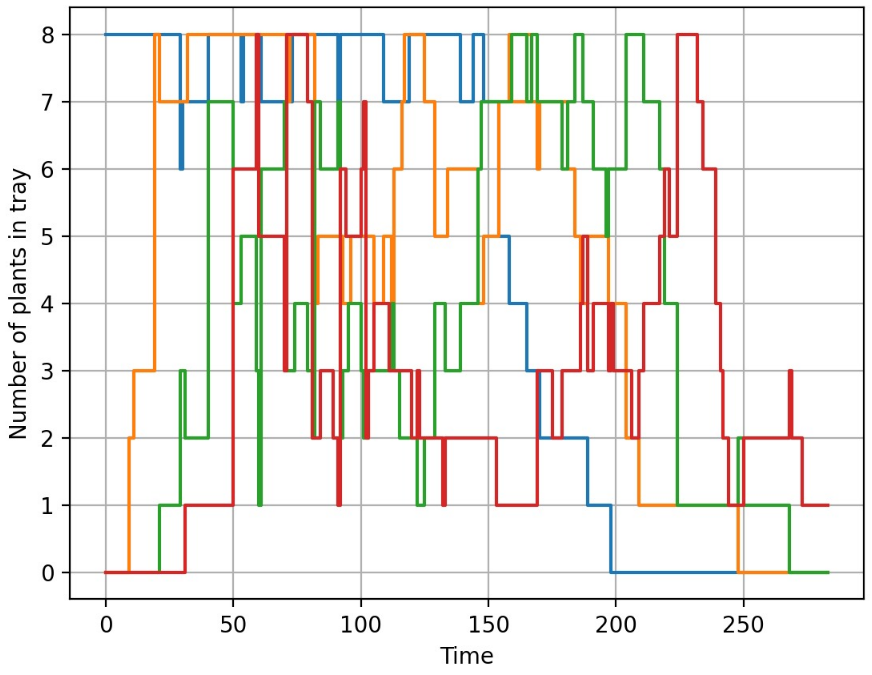


Figure 13: Second configuration - user panel and main results

5.3 Maximum variety configuration

The third and last configuration resides on the maximum variety criteria. Thus we impose a minimum of 7 plants of each type. The result given is a total of 8980kcal in 13 days and 12 hours (324 hours).

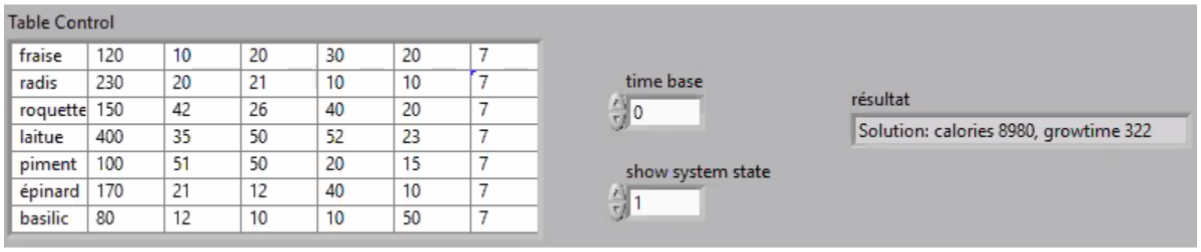


Table Control

fraise	120	10	20	30	20	7
radis	230	20	21	10	10	7
roquette	150	42	26	40	20	7
laitue	400	35	50	52	23	7
piment	100	51	50	20	15	7
épinard	170	21	12	40	10	7
basilic	80	12	10	10	50	7

time base: 0

show system state: 1

résultat: Solution: calories 8980, growtime 322

Figure 14: Third configuration - user panel and main results

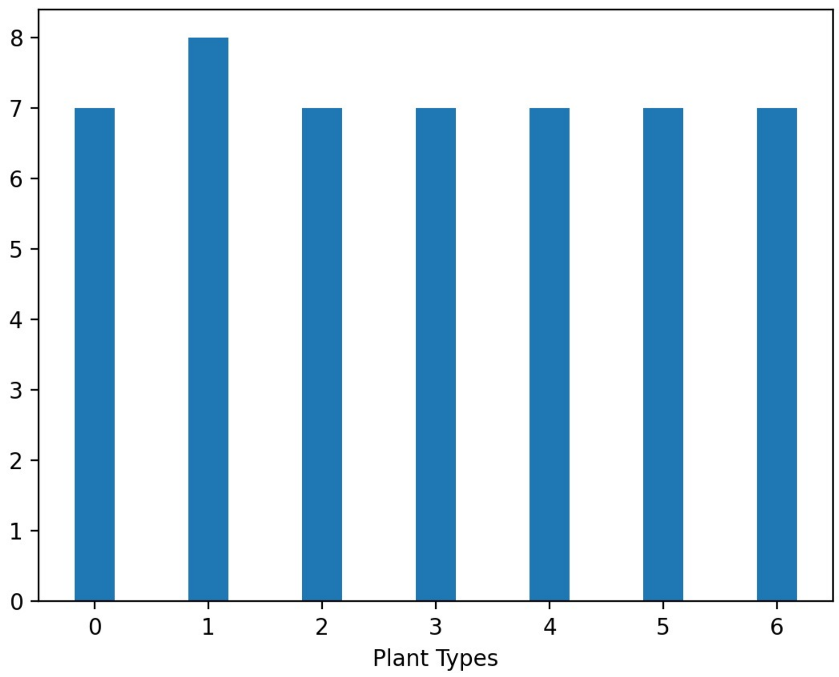


Figure 15: Third configuration - user panel and main results

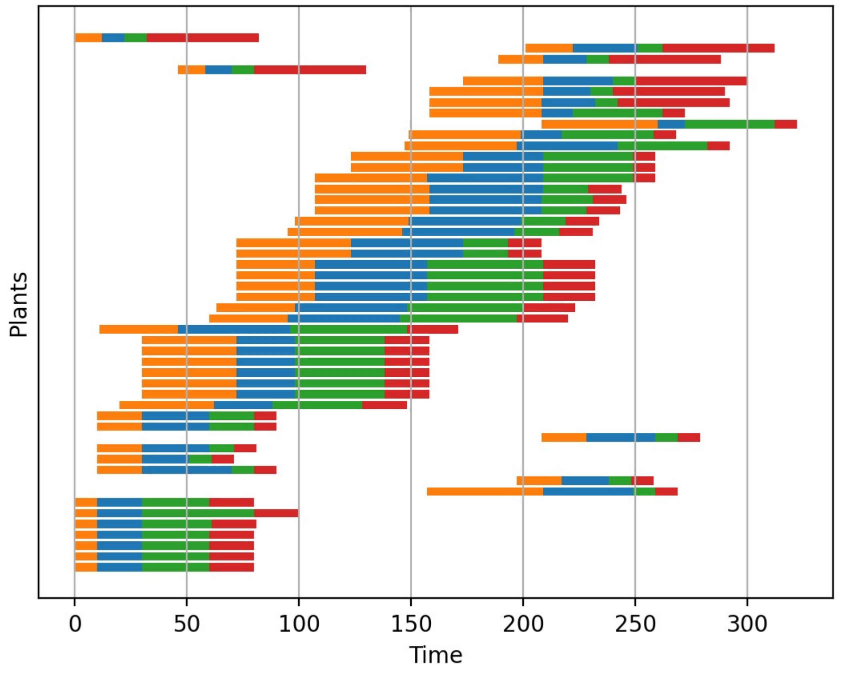


Figure 16: Third configuration - user panel and main results

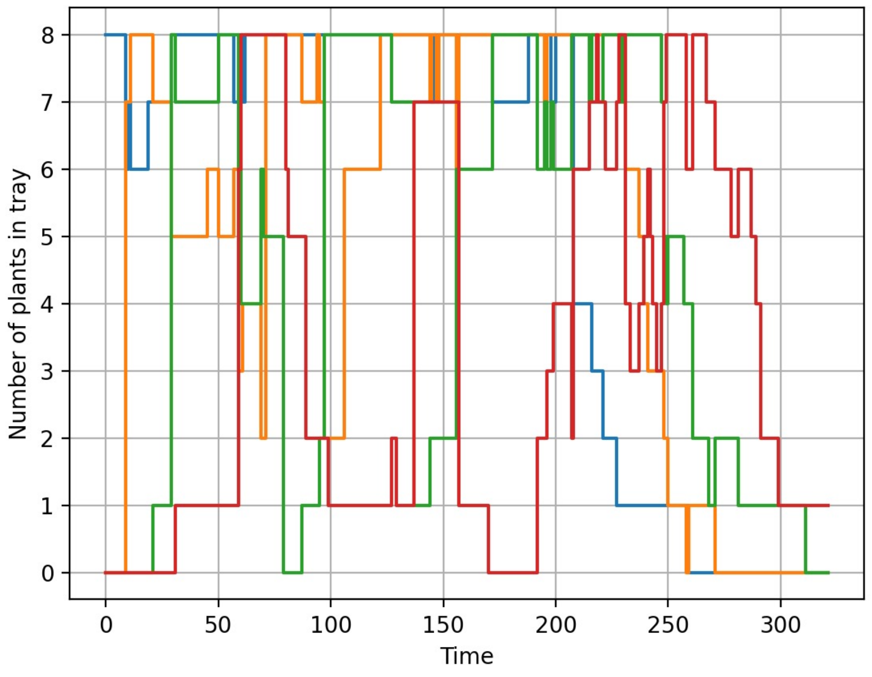


Figure 17: Third configuration - user panel and main results

6 conclusion

We could obtain a time schedule of the system, allowing to follow the plants movements thanks to the data that our program gives to the system. Nevertheless this is just a first step in the way of a further development. Indeed our system is based only on general data and may not represent fully the reality. Also, a big improvement to be able to "read" the state of growth of each plant and adjust the scheduling progressively while the system is progressing. Our program can not be relaunched in the middle of a cycle, so its structure has to be modified if we want it to adjust it self while the running. Another point to work on concerns the transition of a plant "above" a shelf that the plant doesn't need. In our program the plant has to go threw each shelf from 1 to 4, even it does not stay in one. Without a restriction it is likely to think that we could get a non negligible gain of time of a cycle. We think that a good first step was done forward the automatising and scheduling of the system. But good improvements can still be done to get a better optimisation.

7 Acknowledgements

We would like to thank GrowBotHub team for their kind welcome during this past semester. And more specifically Victoria Letertre and Léa Schmidt for their help in determining users and plants data.

This project would have been impossible to realise without the help of our supervisor Prof. Colin Jones, from EPFL Automatic Control Laboratory. We are grateful for the time taken to help us and the precious indications that helped us moving forward during the semester.

8 Annexe

8.1 code matlab

8.1.1 simulation environment

```
clear all;
%% Bachelor Projet - scheduling of vegetables growth -
%matrix containing moving hours: the ligh corresponds to
   the number of the
%plant. The rows represent the hours of entry in the
   shelves and the harvest hour

Plantschange=[[0, 10, 30, 60, 80]; [0, 10, 30, 60, 80];
   [0, 10, 30, 60, 80]; [0, 10, 30, 60, 80]; [0, 10, 30,
   61, 81]; [0, 10, 30, 80, 100]; [0, 10, 30, 60, 80];
   [157, 209, 249, 259, 269]; [197, 217, 238, 248, 258];
   [10, 30, 70, 80, 90]; [10, 30, 51, 61, 71]; [10, 30,
   60, 71, 81]; [208, 228, 259, 269, 279]; [10, 30, 60,
   80, 90]; [10, 30, 60, 80, 90]; [20, 62, 88, 128, 148];
    [30, 72, 98, 138, 158]; [30, 72, 98, 138, 158]; [30,
   72, 98, 138, 158]; [30, 72, 98, 138, 158]; [30, 72,
   98, 138, 158]; [30, 72, 98, 138, 158]; [11, 46, 96,
   148, 171]; [60, 95, 145, 197, 220]; [63, 98, 148, 200,
    223]; [72, 107, 157, 209, 232]; [72, 107, 157, 209,
   232]; [72, 107, 157, 209, 232]; [72, 107, 157, 209,
   232]; [72, 123, 173, 193, 208]; [72, 123, 173, 193,
   208]; [95, 146, 196, 216, 231]; [98, 149, 199, 219,
   234]; [107, 158, 208, 228, 243]; [107, 158, 208, 231,
   246]; [107, 158, 209, 229, 244]; [107, 157, 209, 249,
   259]; [123, 173, 209, 249, 259]; [123, 173, 209, 249,
   259]; [147, 197, 242, 282, 292]; [149, 199, 217, 258,
   268]; [208, 260, 272, 312, 322]; [158, 208, 222, 262,
   272]; [158, 208, 232, 242, 292]; [158, 209, 230, 240,
   290]; [173, 209, 240, 250, 300]; [46, 58, 70, 80,
   130]; [189, 209, 228, 238, 288]; [201, 222, 251, 262,
   312]; [0, 12, 22, 32, 82]; ];
```

```
%matrix containing the type of the plant: plants from 1
   to 7 are of type 0, in this case the strawberries.
 Plantsnames=[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
    1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4,
     4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6,
    6];
 Plantsnames=Plantsnames+1;
 maxplantschange=max(Plantschange);
 %detection of the cycle time
 time=maxplantschange(5);
 size_vector=size(Plantschange);
 %detection of the number of plants
 number_of_plants=size_vector(1);
%% definition of the shelves
% 8 spots per shelf
trails=zeros(4,8);


%% Definition of the types of plants
% name_type = [ name, calories , time_shelf_1 ,
   time_shelf_2 ,
% time_shelf_3 , time_shelf_4 , calories_per_plant ] time
   is mesured in hours
Data_plants(1,:)={"fraise",120,10,20,30,20,};%strawberry
Data_plants(2,:)={"radis",230,20,21,10,10,};%radish
Data_plants(3,:)={"roquette",150,42,26,40,20,};%arugula
Data_plants(4,:)={"laitue",400,35,50,52,23,}; %letuce
Data_plants(5,:)={"piment",100,51,50,20,15,}; %pepper
Data_plants(6,:)={"epinard",170,21,12,40,10,}; %spinach
Data_plants(7,:)={"basilic",80,12,10,10,50,}; %basil

%% Definition of system physics
hour_check=0; % Boolean variableused to verify if the
   prescripted time between two loops.
base_de_temps = 0; % 0 : instanteanous calculus , 1 : 1
   second = 1 hour , 2 : 2 seconds = 1 hour etc..
```

```
show_etat_sys_auto = 1; % number of hours at the end of
   which the file gives a summary of the systems state.
%% number of plants of each type employed in the system.
etat_plante = zeros(number_of_plants,4); %state table of
   all the plants
%column 1 : trails
%column 2 : location
%column 3 : completion of the current grow stage
%column 4 : completion of the global growing

%% Data initialisation
calories_recolte=0;
temps_ecoule=0;
compteur=1;
%% loops of the program
while (temps_ecoule<time+1)
t1=fix(clock);
%harvest


%research of the plant ready to be harvested; free the
   spot if the harvest takes place.
    i=1;
    while i<number_of_plants+1
          if(temps_ecoule==Plantschange(i,5))
            calories_recolte=calories_recolte+Data_plants
               {Plantsnames(i),2};
            trails(4,etat_plante(i,2))=0;
            etat_plante(i,1)=-1;
            etat_plante(i,2)=-1;
            etat_plante(i,3)=100;
            etat_plante(i,4)=100;

          end
        i=i+1;
    end
```

```

%Shelf 4
%research of the plant reasy to enter shelf 4
%checks if a spot in shelf 4 is avaible and if so frees a
    spot in shelf 3
i=1;
while i<number_of_plants+1
        if(temps_ecoule==Plantschange(i,4))
            i_etagere=1;
            while i_etagere<9
                if trails(4,i_etagere)== 0
                    trails(3,etat_plante(i,2))=0;
                    trails(4,i_etagere)= 1;
                    etat_plante(i,1)=4;
                    etat_plante(i,2)=i_etagere;
                    i_etagere=9;
                end
                i_etagere=i_etagere+1;
            end
        end
        i=i+1;
end

%Shelf 3
%research of the plant reasy to enter shelf 3
%checks if a spot in shelf 3 is avaible and if so frees a
    spot in shelf 2

i=1;
while i<number_of_plants+1
        if(temps_ecoule==Plantschange(i,3))
            i_etagere=1;
            while i_etagere<9
                if trails(3,i_etagere)== 0
                    trails(2,etat_plante(i,2))=0;
                    trails(3,i_etagere)= 1;
```

```
                   etat_plante(i,1)=3;
                   etat_plante(i,2)=i_etagere;
                   i_etagere=9;
                end
                i_etagere=i_etagere+1;
            end
        end
        i=i+1;
end

%Shelf 2
%research of the plant reasy to enter shelf 2
%checks if a spot in shelf 2 is avaible and if so frees a spot in shelf 1
i=1;
while i<number_of_plants+1
        if(temps_ecoule==Plantschange(i,2))
            i_etagere=1;
            while i_etagere<9
                if trails(2,i_etagere)== 0
                   trails(1,etat_plante(i,2))=0;
                   trails(2,i_etagere)= 1;
                   etat_plante(i,1)=2;
                   etat_plante(i,2)=i_etagere;
                   i_etagere=9;
                end
                i_etagere=i_etagere+1;
            end
        end
      i=i+1;
end

%Shelf 1
%research of the plant reasy to enter shelf 1
%checks if a spot in shelf 1 is avaible.
i=1;
```

```
while i<number_of_plants+1
      if (temps_ecoule==Plantschange(i,1))
            i_etagere=1;
            while i_etagere<9
                 if trails(1,i_etagere)== 0
                       trails(1,i_etagere)= 1;
                       etat_plante(i,1)=1;
                       etat_plante(i,2)=i_etagere;
                      i_etagere=9;
                 end
                 i_etagere=i_etagere+1;
            end
      end
       i=i+1;
end










%Upgrade of the growth value
%Computes the growth percentage on the overall cycle and
   on also
%on the the one of the current stage.
%Only works for plants allready on a shelf.

i=1;
while i<number_of_plants+1
     if (Plantschange(i,1)<temps_ecoule) && (Plantschange(
        i,5)>temps_ecoule)
          etat_plante(i,4)=((temps_ecoule-Plantschange(i,1)
             )/(Plantschange(i,5)-Plantschange(i,1))*100);
        etat_plante(i,3)=((temps_ecoule-Plantschange(i,etat_plante(i,1)))/(Plantschange(i,etat_plante(i,1)+1)-Plantschange(i,etat_plante(i,1)))*100);
    end

i=i+1;
end

%loop allowing the the program to be used in real time or in the chosed time basis.

if (base_de_temps ~= 0)
while (hour_check==0)
    t2=fix(clock);
    time_elapsed=etime(t2,t1);
    if time_elapsed>base_de_temps
        hour_check=1;
    end
end
end
hour_check=0;
temps_ecoule=temps_ecoule+1;
end
```

8.1.2 approximation de la fonction objective

```
%% Approximation of the non-linear objective function

clear all;
Data_plants(1,:)={"fraise",120,10,20,30,20,}; %strawberry
Data_plants(2,:)={"radis",230,20,21,10,10,}; %radish
Data_plants(3,:)={"roquette",150,42,26,40,20,}; %arugula
Data_plants(4,:)={"laitue",400,35,50,52,23,}; %letuce
Data_plants(5,:)={"piment",100,51,50,20,15,}; %pepper
Data_plants(6,:)={"epinard",170,21,12,40,10,}; %spinach
Data_plants(7,:)={"basilic",80,12,10,10,50,}; %basil
[ligne,colonne]=size(Data_plants);
division=[];
sol=[];
%establish the descreasing order of the plants in
   function if the calries/time ratio.
for n= 1:ligne
    division(n)=Data_plants{n,2}/(Data_plants{n,3}+
       Data_plants{n,4}+Data_plants{n,5}+Data_plants{n,6})
       ;
end
%looking for a and b to obtain the same order than the
   non linear function
[divmax,idivmax]=max(division);
for a = 0 : 1 : 10
     for b = 0: 1 : 10
          for n= 1:ligne
    soustraction(n)=a*Data_plants{n,2}-b*(Data_plants{n
       ,3}+Data_plants{n,4}+Data_plants{n,5}+Data_plants{n
       ,6});
          end
          [soumax,isoumax]=max(soustraction);
          if (isoumax==idivmax)
              sol=[sol,a,b];
          end
     end
end
```

8.1.3 VI labview et explication

The control VI (principal VI) is the VI which allows the user to control the different programs and lauch the scheduling. The entry data is the data given by the user: type of plant, minimum number of plant by type, definition of time basis and the time frequency of the summary given.

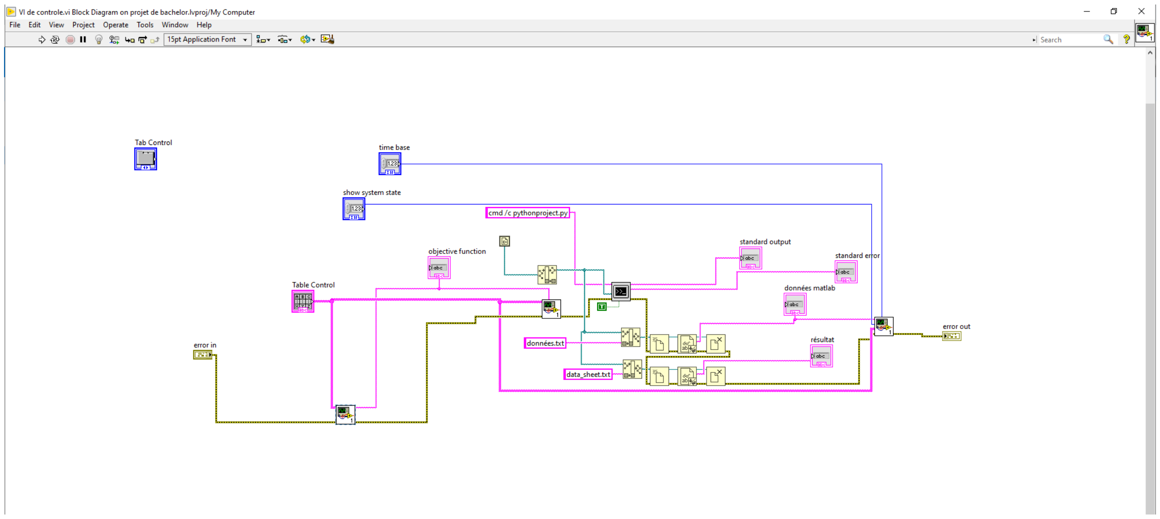
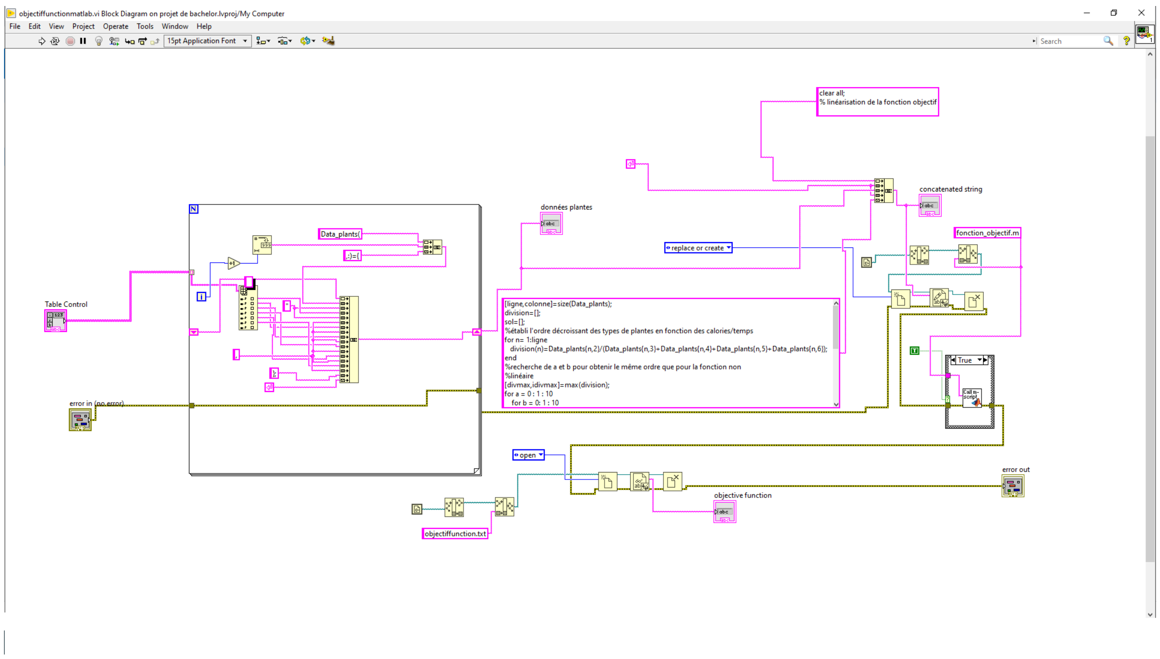


Figure 18: Block diagram of Control VI

The objective function VI is in charge of finding the variables a and b to linearise optimally the objective function defined by calories/time. This linearization is operated by Matlab via a comparison of the different types obtained by the division and subtraction. (The order is based on "calories/time" for the division and $(a \cdot calories - b \cdot time)$ for the subtraction. In entry it receives the table of the types of plants given by the user. In exit, it gives the line of command defining the objective function for the python code.

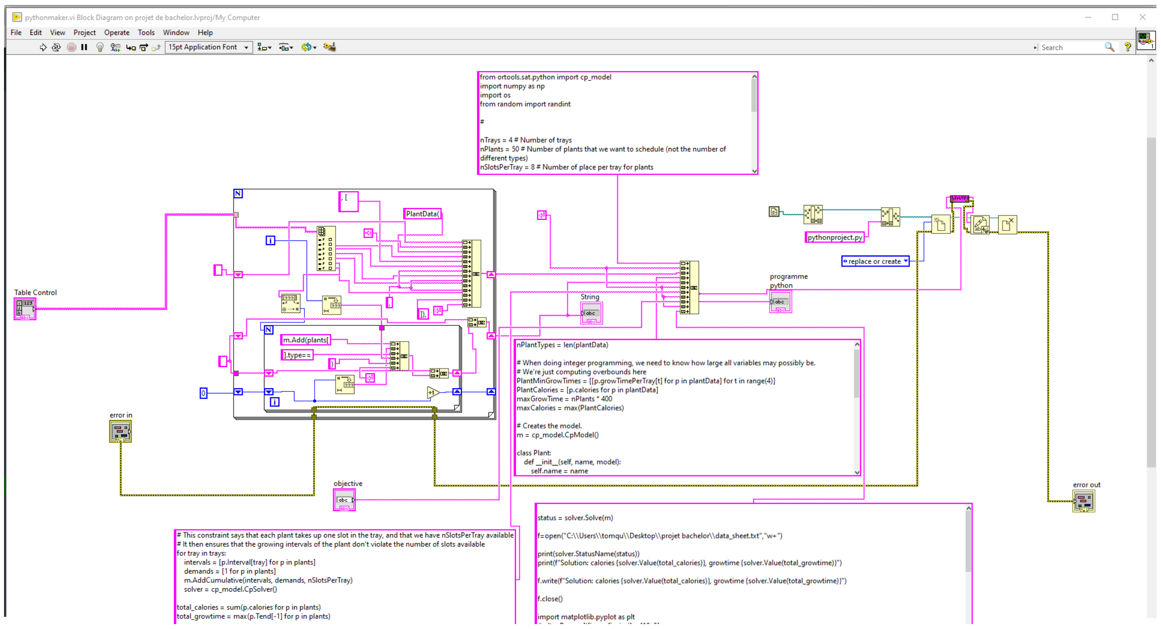


```
clear all;
% linéarisation de la fonction objectif
```

```
[ligne,colonne]=size(Data_plants);
division=[];
sol=[];
%établi l'ordre décroissant des types de plantes en fonction des calories/temps
for n= 1:ligne
   division(n)=Data_plants{n,2}/(Data_plants{n,3}+Data_plants{n,4}+Data_plants{n,5}+Data_plants{n,6});
end
%recherche de a et b pour obtenir le même ordre que pour la fonction non
%linéaire
[divmax,idivmax]=max(division);
for a = 0 : 1 : 10
    for b = 0: 1 : 10
```

Figure 19: Block diagram of objecctiffunction VI

Pythonmaker VI is in charge of re-writing the Python code with the user data. Thus it modifies the types of each plants and the "minimum of plants per type" constraint. Also, it changes the objective function accordingly to the first program. Thus in entry it needs to have the tables of types of plants given by the user and function given by the "objectiffunction" VI. We had to force the writing in UTF-8 to launch properly the python code.



```
from ortools.sat.python import cp_model
import numpy as np
import os
from random import randint

#

nTrays = 4 # Number of trays
nPlants = 50 # Number of plants that we want to schedule (not the number of
different types)
nSlotsPerTray = 8 # Number of place per tray for plants
```

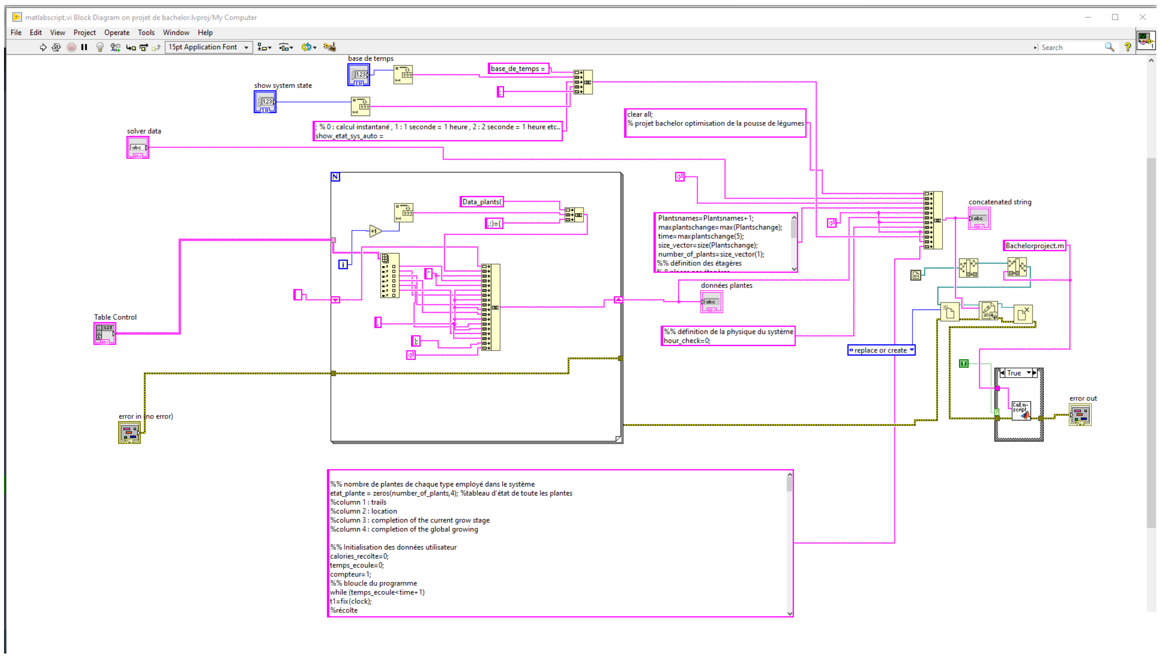
```
nPlantTypes = len(plantData)

# When doing integer programming, we need to know how large all variables may possibly be.
# We're just computing overbounds here
PlantMinGrowTimes = [[p.growTimePerTray[t] for p in plantData] for t in range(4)]
PlantCalories = [p.calories for p in plantData]
maxGrowTime = nPlants * 400
maxCalories = max(PlantCalories)

# Creates the model.
m = cp_model.CpModel()

class Plant:
   def __init__(self, name, model):
      self.name = name
```

```
# This constraint says that each plant takes up one slot in the tray, and that we have nSlotsPerTray available
# It then ensures that the growing intervals of the plant don't violate the number of slots available
for tray in trays:
   intervals = [p.Interval[tray] for p in plants]
   demands = [1 for p in plants]
   m.AddCumulative(intervals, demands, nSlotsPerTray)
   solver = cp_model.CpSolver()

total_calories = sum(p.calories for p in plants)
total_growtime = max(p.Tend[-1] for p in plants)
```

```

status = solver.Solve(m)

f=open("C:\\Users\\tomqu\\Desktop\\projet bachelor\\data_sheet.txt","w+")

print(solver.StatusName(status))
print(f"Solution: calories {solver.Value(total_calories)}, growtime {solver.Value(total_growtime)}")

f.write(f"Solution: calories {solver.Value(total_calories)}, growtime {solver.Value(total_growtime)}")

f.close()

import matplotlib.pyplot as plt
```

Figure 20: Block diagram of Pythonmaker VI

After the Python code being executed and the optimisation launched, we get a txt file named "données" which contains the time schedule of plants movements. We send it together with the "type of plants" table to the last VI: Matlabscript VI which the goal is to write the script of Matlab code used to simulate the system. Finally, it gives an exit file where the movements are registered in the form of a txt file, which will be sent to the micro-controller.



```
base_de_temps =
```

```
;
```

```
; % 0 : calcul instantané , 1 : 1 seconde = 1 heure , 2 : 2 seconde = 1 heure etc..
show_etat_sys_auto =
```

```
clear all;
% projet bachelor optimisation de la pousse de légumes
```

```
Data_plants(
```

```
,:)={
```

```
Plantsnames=Plantsnames+1;
maxplantschange=max(Plantschange);
time=maxplantschange(5);
size_vector=size(Plantschange);
number_of_plants=size_vector(1);
%% définition des étagères
```

```
%% définition de la physique du système
hour_check=0;
```

```
%% nombre de plantes de chaque type employé dans le système
etat_plante = zeros(number_of_plants,4); %tableau d'état de toute les plantes
%column 1 : trails
%column 2 : location
%column 3 : completion of the current grow stage
%column 4 : completion of the global growing

%% Initialisation des données utilisateur
calories_recolte=0;
temps_ecoule=0;
compteur=1;
%% bloucle du programme
while (temps_ecoule<time+1)
t1=fix(clock);
%récolte
```

Figure 21: Block diagram of Matlabscript VI