FarmBot

Délivrable 2

Farmbot : un potager automatisé

Yoann d'Erneville - Thomas Legris - Théo Robin

Team Farmbot

UNIVERSITÉ DE RENNES 1

esir ECOLE SUPERIEURE D'INGENIEURS DE RENNES

Sommaire

Introduction :

Après un premier délivrable qui présentait de façon globale notre équipe et le projet que nous allons réalisé, ce deuxième délivrable se veut plus spécifique notamment au niveau logiciel. En effet dans celui-ci, nous axerons notre développement sur l'architecture du Farmbot et le fonctionnement du point de vue informatique de celui-ci. Après cette étude, nous expliquerons les différentes solutions que nous allons mettre en place pour répondre à nos objectifs.

I - Cahier des charges

Dans cette partie, nous avons réalisé un tableau récapitulant notre cahier des charges. Dans la partie "Nos modifications", nous affinerons ces différents objectifs en fonction de ce qu'il nous semble réalisable notamment du point de vue logiciel.

Niveau d'importance	Description des objectifs
Haute importance	- Rendre autonome le Farmbot grâce aux énergies renouvelables. Utilisation de l'énergie solaire pour l'alimentation électrique et la récupération d'eau de pluie pour l'arrosage. - Modification et amélioration du logiciel avec l'ajout de conseils et suggestions en fonction de la saison
Moyenne importance	- Optimisation de la gestion de l'eau en installant une station météo. Modification du logiciel pour indiquer à l'utilisateur les différentes prévisions (soleil et précipitations).
Basse importance	- Gestion efficace des eaux pluviales, par exemple : éviter que l'eau stagne. - Création d'un outil permettant de retourner la terre dans le bac.

Figure 1 - Tableau récapitulatif des objectifs

II - Assemblage

Lors de la séance du 10 novembre 2017, nous avons participé au montage du Farmbot. Le Farmbot était déjà prémonté grâce au travail de M. Johann Bourcier. Nous avons réalisé plusieurs tâches lors de cette séance dont notamment l'assemblage des différentes têtes et de la partie électronique (Raspberry Pi, Arduino). Pour réaliser toutes ces opérations, nous avons utilisés la documentation fournie par les constructeurs du Farmbot.

La première tâche a été l'assemblage des têtes. Les différentes têtes viennent se fixer sur la tête universelle. Cette connexion est réalisée grâce à un système d'aimant. La connexion électrique se fait, elle, grâce à des vis à ressort situées sur la tête universelle.
Les différentes têtes que nous avons montées sont :

- la tête "seeder" qui gère la plantation des graines
- la tête avec le capteur qui se met directement dans le sol
- la tête "weeder" pour le désherbage
- la tête "water" permettant l'arrosage

Nous avons également positionné la caméra sur le bras du Farmbot.

La seconde tâche a été la réalisation du boitier électronique et la mise en place de celui-ci sur le Farmbot. Pour cela, nous avons dû tout d'abord fixer les différents composants électroniques au boitier puis ensuite installer le boîtier sur le rail prévu à cette effet. L'installation se fait grâce à des écrous spéciaux appelés "tee nut". Le cablâge de l'arduino doit être fait de façon précise pour ensuite permettre le fonctionnement adéquat du Farmbot. L'ordre est le suivant : caméra USB, câbles UTM, cables moteurs, encodeurs, périphériques et câble d'alimentation.

Figure 2 - Montage des têtes

Figure 3 - Boitier électronique

Un travail important est encore nécessaire pour que l'assemblage du Farmbot soit réalisé dans son ensemble. La réalisation du socle, l'alimentation et le test du fonctionnement des composants électroniques sont les principales tâches qu'il reste à faire.

III - Architecture logicielle

Le Farmbot est construit en quatre principaux blocs qui sont chacuns reliés entre eux.

Tout d'abord la "Farmbot Web application" [1], qui est l'interface permettant à l'utilisateur d'interagir de différentes façons avec le robot. Ensuite, il y a la partie Raspberry [2], directement connecté à la web-App. Elle permet de synchroniser chaque demande, chaque nouvelle séquence et de remonter les informations jusqu'à l'utilisateur. Ce bloc est relié à l'arduino [3], ce micro-contrôleur permet de contrôler la partie matérielle et de retourner les opérations.

* Les chiffres réfèrent au schéma de la page suivante

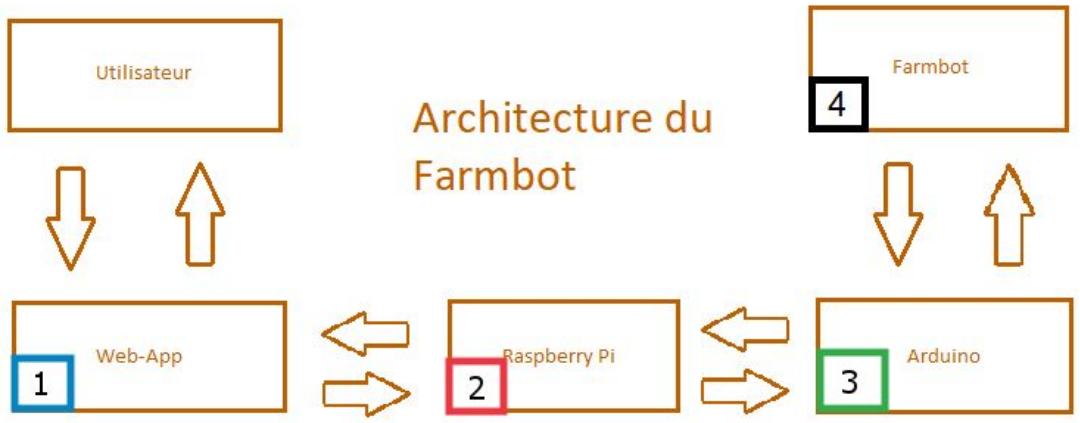


Figure 5 : Architecture simplifiée du Farmbot

La structure du Farmbot est composé de 4 grandes parties, la dernière est la partie Farmbot. Ce terme défini en faites le robot en lui même, toutes la partie Hardware connecté à l'arduino.
Nous allons étudier séparément le fonctionnement de chaque blocs.

1. L'application Web

Ce que les créateurs du Farmbot appelle la "Web-App" est en réalité l'outil qui sert d'interface entre l'utilisateur et le Farmbot. Cette interface est accessible pour tous les utilisateurs sur le web, sur leurs smartphones et également sur tablette . La Web-App est de base hébergé sur my.farmbot.io. Cependant, il est possible de l'héberger localement.

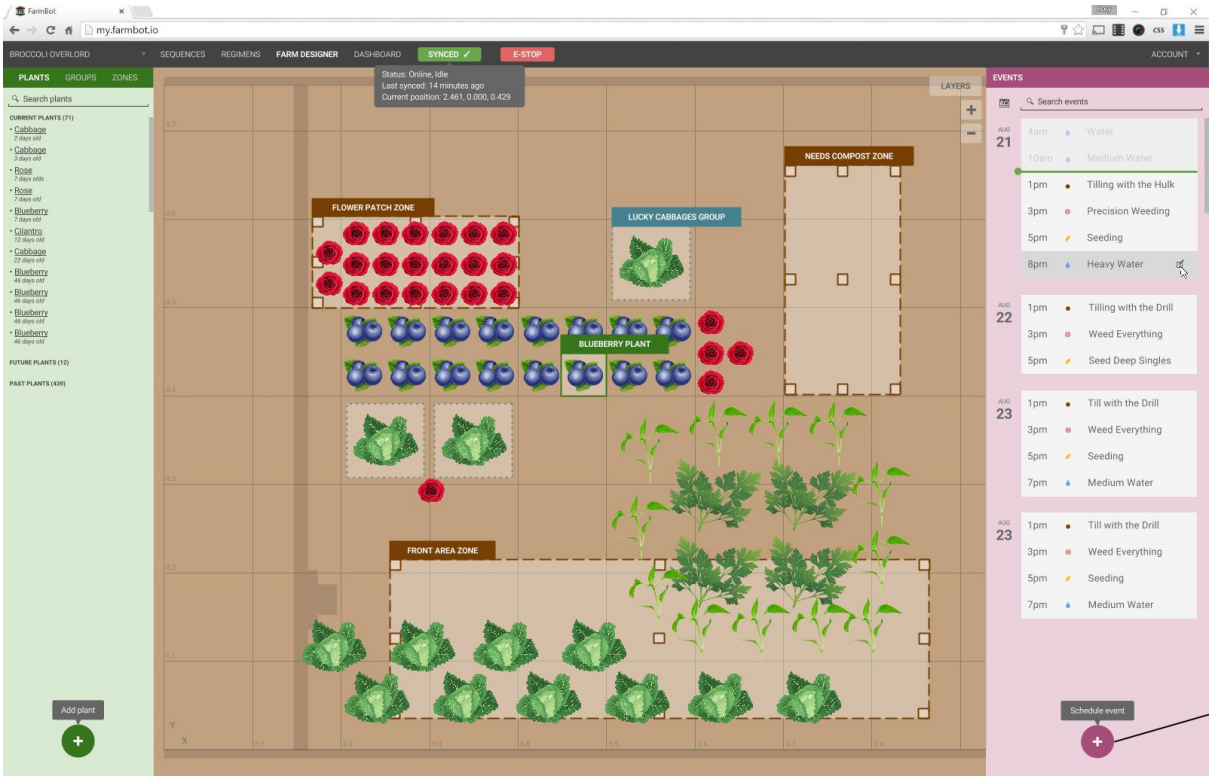


Figure 4 : Aspect de la Web-app

Cet outil permet de réaliser des tâches tels que le placement des différentes plantes grâce à un système de glisser-déposer. L'utilisateur a un plan de son jardin et une liste des plantes et légumes à sa disposition. Il lui suffit de glisser la plante qu'il souhaite planter à l'emplacement souhaité. Une fois l'application synchronisé avec le Farmbot, celui-ci va automatiquement planter la graine choisie au bon endroit.
La réalisation d'événement possède des fréquences définies par l'utilisateur. Par exemple, l'utilisateur créé un évènement d'irrigation de son jardin et décide de le mettre en place toutes les semaines le même jour et à la même heure.

L'utilisateur peut contrôler manuellement le bras de Farmbot en direct et réaliser les mouvement de déplacement (haut, bas, gauche, droit …) .

L'application web permet d'afficher les informations sur le système et l'état de la connexion entre les différentes parties du Farmbot. Par ailleurs, elle sert à initialiser le robot Farmbot.
On peut définir où sont placées les différentes têtes pour qu'ensuite le bras du Farmbot puisse les utiliser. Une dernière possibilité pour l'utilisateur et de configurer la caméra ainsi que définir la sensibilité à la détection des mauvaises herbes.

2. La Raspberry Pi

La Raspberry Pi est un élément essentiel au fonctionnement du Farmbot. En effet, il fait le lien entre la Web-App et la partie Arduino. C'est lui le cerveau du robot. La connexion

entre l'application web et la Raspberry Pi se fait en Ethernet ou en Wifi. Dans les deux cas une passerelle MQTT est utilisée. Le serveur MQTT est appelé "*broker*". C'est lui qui va collecter les informations. Les clients MQTT, sont abonnés à des topics. Le serveur peut alors envoyer des informations à tous les clients connectés à un même topic.
Cette connexion permet de transférer les commandes désirées par l'utilisateur lorsqu'il réalise par exemple une séquence de plantation ainsi que de faire remonter des informations sur l'état des capteurs.
La Raspberry Pi est connectée au contrôleur Arduino en USB pour lui transmettre les différentes commandes et recevoir des données tels que celles fournies par le capteur de sol.

3. Le contrôleur Arduino

Le contrôleur Arduino ATmega réalise le lien avec la partie mécanique du Farmbot. En fonction des commandes reçues par la Raspberry Pi, il commande les différentes parties du Farmbot. Les moteurs sont dirigés grâce à la carte RAMPS qui est branchée sur l'arduino. Cette carte permet d'interfacer plusieurs fonctionnalités comme la commande de plusieurs moteurs. Le contrôle se fait par envoi d'impulsions électriques, l'Arduino interagit alors avec les moteurs pour déplacer le bras et réaliser l'ensemble des commandes . Pour ce qui est du contrôle et du choix des têtes, cela est réalisé grâce à l'écriture sur les broches (PINs) correspondantes. L'Arduino récolte également les données de ses capteurs pour les transmettre à la Raspberry qui, ensuite, fait remonter les informations à la Web-App via la passerelle MQTT.

III - Fonctionnement logiciel

1. Introduction au fonctionnement du robot

Comme expliqué précédemment, il y a 5 grandes parties dans ce système. Voici le schéma de l'organisation du Farmbot avec les différentes connexions entres les composants du Farmbot.

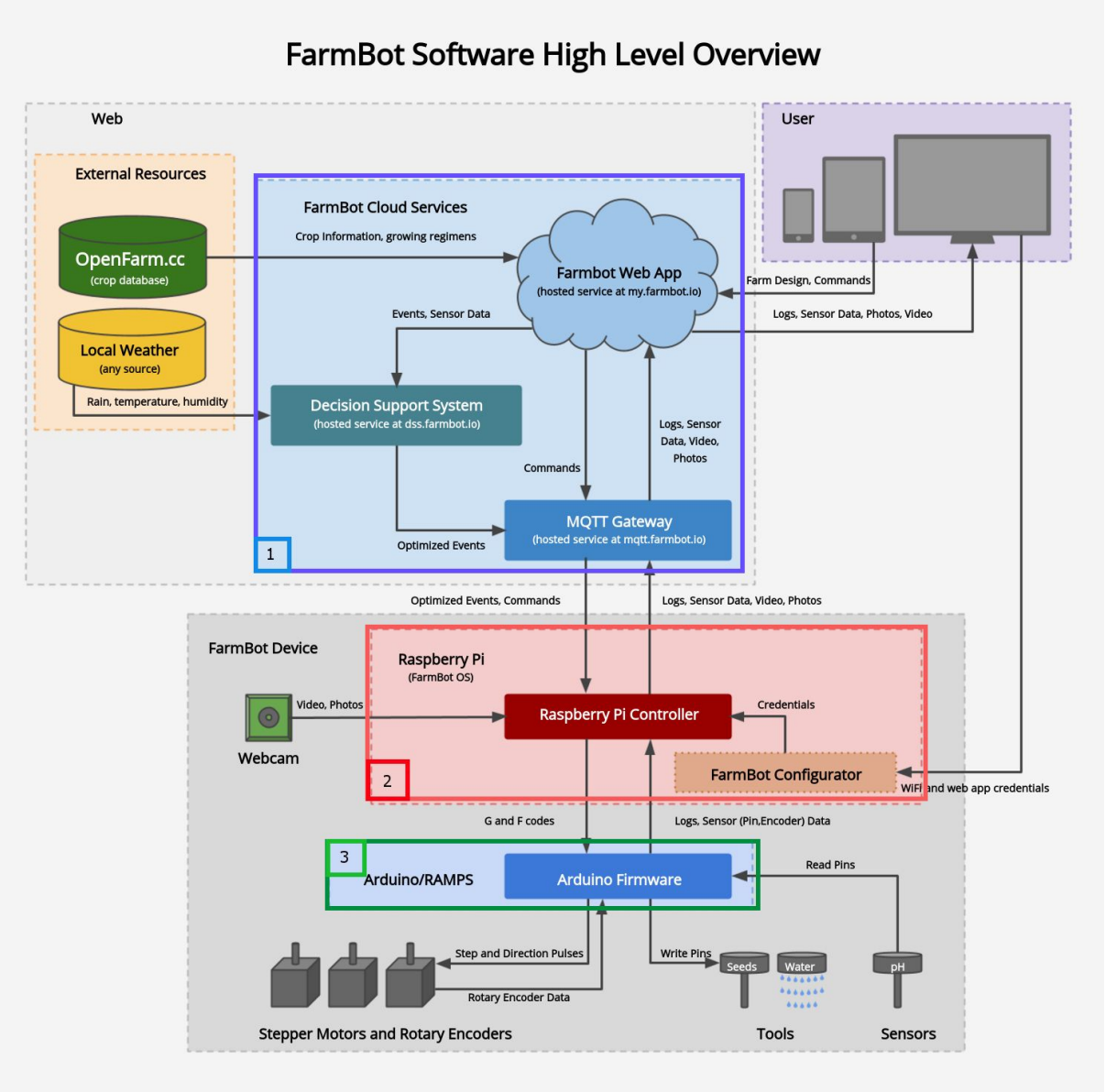


Figure 6: Schéma des différents composants et de leurs interactions
Source : https://software.farmbot.io/docs

Chaque partie du programme est différente en termes de software, ce qui veut dire que nous devons étudier chacune d'elles afin de pouvoir correctement modifier le Farmbot. Tout le code se trouve sur le GitHub du farmbot. Nous allons donc nous y intéresser.

2. La plateforme Web

Celle-ci permet à l'utilisateur d'interagir avec le Farmbot. Elle est codée en Ruby, qui est un langage objet open-source.
Une fois le code récupéré, l'identification des différentes fonctions commence.
La Web-App comporte plusieurs fonctionnalités, de l'identification des clients à la création des légumes (nom de la plante, position dans le bac).

Le protocole MQTT est un intermédiaire entre la web app et la raspberry, c'est à dire entre le browser et le Farmbot, il est utilisé pour permettre de recevoir les données (état des capteurs et des moteurs) mais aussi pour envoyer les requêtes des clients au Farmbot.

```
bot
  .connect()
  .then(function(bot){
    console.log("Bot online!");
    return bot.emergencyStop(); // You can chain commands.
  })
  .then(function(bot){
    console.log("Bot has stopped!");
  })
  .catch(function(error) {
    console.log("Something went wrong :(");
  });
```

Figure 7 : client MQTT : envoi de commande au Farmbot.

```
/** Move gantry to an absolute point. */
moveAbsolute(args: { x: number, y: number, z: number, speed?: number }) {
  let { x, y, z, speed } = args;
  speed = speed || Farmbot.defaults.speed;
  return this.send(rpcRequest([
    {
      kind: "move_absolute",
      args: {
        location: coordinate(x, y, z),
        offset: coordinate(0, 0, 0),
        speed
      }
    }
  ]));
}
```

Figure 8 : Code JS permettant de bouger le bras du farmbot.

Il y a plusieurs types de code dans la Web App, le Ruby comme on l'a vu mais aussi le code de la page Web : HTML5 afin de générer la page web et le CSS pour gérer l'aspect graphique, comme les boutons ou bien les couleurs. Ce n'est pas exactement du CSS, mais

du SCSS, cela permet de gérer le style de la web App. Le SCSS permet d'utiliser plus de fonctionnalités que le CSS classique comme l'ajout de variables.

```
*::-webkit-scrollbar {
  width: 4px;
  height: 4px;
  background-color: $light_gray;
}

```

Figure 9 : Exemple de code SCSS

3. Partie Raspberry

3.1. Communication Raspberry - Arduino

La raspberry reçoit les codes transférés via la passerelle MQTT, elle doit alors les interpréter et les transmettre sous forme de code F et G à l'arduino. Pour cela, elle utilise les ports GPIO classique. L'équipe Farmbot avait ajouté une surcouche 'serial' en 2015, permettant la gestion des évènements sur les I/O, mais elle est finalement revenue en arrière en la supprimant.

3.2. Communication Raspberry-Web App

La Raspberry Pi communique avec le Farmbot via un style d'architecture, REST qui repose sur le protocole HTTP. La Raspberry Pi accède à la Web-App grâce à son URL unique pour avoir accès à diverses opérations comme GET, POST PUT, DELETE. Les formats d'échanges sont de type JSON. Ici, nous voulons savoir comment la Raspberry Pi communique avec la passerelle MQTT. Le logiciel de contrôle de l'ensemble du Farmbot réside dans la carte SD de la Raspberry Pi. Il est développée en *Elixir* et utilise le Projet *NERVES*.

Ce projet NERVES est une collection de projets permettant d'implémenter des systèmes embarqué utilisant Erlang et Elixir. L'idée derrière ce projet est d'utiliser les outils Elixir ou Erlang pour créer une application qui sera combiné avec un petit système qui utilise Linux (ici notre Raspberry Pi) et ce dernier lancera automatiquement l'application. Pour cela on utilise des méthodes *mix* en Elixir . Ce dernier est présent dans presque tous les codes.

Elixir est un langage de programmation qui fonctionne sur la machine virtuelle Erlang (BEAM). C'est un langage dynamique et fonctionnel utilisé pour les sites web

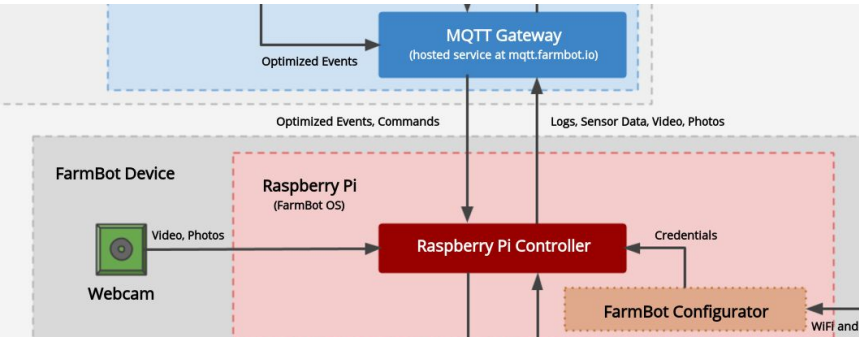


Figure 10 : Partie communication RPI et MQTT Gateway

Le schéma précédent (figure 10) correspond à la nouvelle version du Farmbot (V1.3). Cette version n'utilise plus de FarmBot Mesh mais plutôt la passerelle MQTT (MQTT Gateway).

Le serveur MQTT utilisé est hébergé à l'adresse mqtt.farmbot.io. C'est lui qui assure la transmission entre la Web App et la Raspberry Pi. Il joue le rôle d'une passerelle. La connexion avec la raspberry pi est assurée grâce à une connexion TCP. La connexion entre la Web App et le serveur MQTT, elle, se fait via l'utilisation de Websocket.

Celery est un type de message échangé, il permet aux utilisateurs de communiquer avec le farmbot à travers le serveur MQTT. En effet, c'est une façon spéciale de générer des messages JSON. Comme on l'a vu lors de nos cours de Web, les messages sont envoyés avec le format JSON, avec différents paramètres. On peut donc lister les méthodes possibles :

```
export declare type CeleryNode = Nothing | Tool | Coordinate | MoveAbsolute | MoveRelative |
WritePin | ReadPin | Channel | Wait | SendMessage | Execute | If | Sequence | Home |
FindHome | Zero | EmergencyLock | EmergencyUnlock | ReadStatus | Sync | CheckUpdates |
PowerOff | Reboot | TogglePin | Explanation | RpcRequest | RpcOk | RpcError | Calibrate | Pair
| RegisterGpio | UnregisterGpio | ConfigUpdate | FactoryReset | ExecuteScript | SetUserEnv |
TakePhoto | Point | InstallFarmware | UpdateFarmware | RemoveFarmware | ScopeDeclaration
| Identifier | VariableDeclaration | ParameterDeclaration | SetServoAngle |
InstallFirstPartyFarmware;
```

Figure 11 : Liste des méthodes

Les messages échangés sont faites des actions (du type mouvement :"moveAbsolute" par exemple) ou bien simplement des configurations. Le code suivant nous montre comment son déclarées les méthodes listées auparavant. Ici, kind est l'attribut correspondant au nom du message

```
export interface PowerOff {
                                kind: "power_off";
                                args: {};
                                comment?: string | undefined;
                                body?: undefined; }
```

Figure 12 : Exemple de construction d'un message

La configuration du Farmbot utilise des Webpack pour compiler un Projet TypeScript en un serveur Web Statique. Nous remarquons au passage que l'OS de Farmbot est Cryptée par l'algorithme RSA dont nous voyons sa clé privée ici :

28 lines (27 sloc) | 1.64 KB

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAsIgWhfZew/k4wE8hWf82RErhP1fsy8v6NbiR5HY06miqWNuI
F1tKg+CHe+NoyaADUJ+Qfu/LIMvSN7USEHls9mEl5Kx1H27hbPgu+TuqqQTgS2rF
snLjWPTQFigH5KNm5HSoreq2/cXRGdXd9IuIwyVh4AFyvSxI7/GrWlKL6M1kZv39
N56zlDt086L5OKltOGIVJF6uJypsXy02omEH4L/zd2npZKD8Y7kaFfNs5mnWw65I
Wr2tqjF7cp8ESN93ChwM7z4I+Xg+EghsEkHVNS4pdVgE5dyB10P/pNjjS52rs6uJ
D3M/Tigr9wS5T/qqbNmcksVT1aP8rWAys4QnIQIDAQABAoIBADUBR7IFncK+LDoi
CGOba9HpoeSBJAq1PnWu669rhsvzjWKM2DobIS6j1kpup+ISd6xXnO1gVt+ME5zC
c6AatYrs9JHK7of3pRwxEPmo3r9NRYOflajVMkpdh7V/Y49VOOnT1WoTFcrxAK3/
N1vcIb5mlRLLnIYMrAHP0KGYM4Y81dpsoeNFO8odWseE06uHVTbRBIAHbNvRy6zd
LnEidEh9soNNcVDrulXbAFmiPfoO/Bjbp0RYhg/vMhadNernktw9zrXFUQxAAjVR
SC8Svl07Sz6KcwL72Vf3hgzJCzCCpOyU8w8dPAMRXrY4r5niCfIJlXxqH9EiH80h
NmMMaHkCgYEA38FL1iuAI2lh4S33/YMdLpk5SMrSLZJPX/CqppmFfEaNQROFOiAl
1LDP5JbG9W37icf6qJsYsVeO0AMR3YC1CtxXQ7Wzif/TSVo24Ww9iSZTcz7Y6lVo
tIW+iwUyAXE3pnxHXS2j8Zj/Z6vm2B5oYbMy+NlF/mhVkCOXByi4xkcCgYEAyfij
Sqsh4TmJJCPB4/MEd+MrdggPAGVqKLtOggGc2OsQpqsviHFMTgEN60Gu7jeATcQM
61Vi0HLvPjX7DGWw1tvT1BLZvSp+fniME94Z+mk48jBILwXCqTuM0ZXNcU0XjgIA
jLk+E8/I3Sw9eAs5Z95r+R5gtNkT0TcO/C8yk1cCgYEA3LTJnSOjbUqRZY/2QXWG
32P8ATUuRA1BhhzZ9yMPbBobUslybHcxWa5eIdgnwAcQSkOb15wEq0j2cW/Vu2st
KN1Wpk8gHUremkgGQiyGNjY7sj2XsO02LnqODIq/XHTUs796lQpj3/dOVnBVb2/u
/g/Ig3WteNhpLZgtbL5aJBkCgYEAsHtUpEBZQGZ4EV41ZCvLsb6NEXwFL8FuO90/
wpYKKfls+VYIGN93X4nIUdN5OarBsDIpX9GioKZtqxycG78YAQbhIDhAjuz8zyIi
tJGUfZ1IJ0hNKtmLuTjR2aledSx58pqJRG3xcnpT9/9aTvTv2nUeP/ZtZllw2ZWU
wIO1W80CgYEAkJ7LRsAUfwV/xn3aywvE3uwESQwrfBixcl6BGP3idSJYNvTx+SO9
XzLg5hbdCqTst8jNvz6VDvExRRUFNgi38BvO55wo0JLDKOz/uj3xaSDFX7L5KeSV
z+3rvi4ulZHA87Y9MDrrZ1KzMFWHJ9YWFO9Cfua4130eSRD2FJNP28o=
-----END RSA PRIVATE KEY-----
```

Figure 13 : Clé privée RSA de l'OS Farmbot

Les principales choses à connaître en Elixir pour comprendre les codes sont résumés dans le tableau ci-dessous :

def (Param) do … end	Fonction publique qui prend en paramètre Param
defp (Param) do … end	Fonction privée qui prend en paramètre Param
Expression |> Function	Syntaxe qui permet d'introduire l'expression

	à gauche en tant que paramètre dans la fonction à droite du signe
Case … do	Comparatif. Il peut s'utiliser sans mettre de cas initial ou de variable initiale. ex: case {1,2,3} do {4,5,6}
defexception(fields)	Définit une exception
defprotocol(name, do_block)	Définit un protocole

Figure 14 : Aide à la compréhension d' Elixir

Le code source de l'OS écrit en Elixir, utilise un module standard : le *GenServer*. Ce module permet de construire une relation client serveur simple avec un partage commun de ressources. Il fonctionne comme un registre central avec son propre état et sur lequel on peut effectuer des actions asynchrones via des passages de messages.

```
@typedoc """
  State for this GenServer
"""
@type state :: %{
  context: Context.t,
  nerves: nerves,
  tty: binary,
  current: current,
  timeouts: integer,
  status: status,
  initialized: boolean
}
```

Figure 15 : États du GenServer

Avant de démarrer, le serveur a besoin de variables comme le booléen qui renseigne sur l'état du serveur (*initialized*) ou alors un integer pour renseigner sur le temps écoulé (*timeouts : integer*).

```
@doc """
  Starts a UART GenServer
"""
def start_link(%Context{} = ctx, nerves, tty, opts)
when is_pid(nerves) and is_binary(tty) do
  GenServer.start_link(__MODULE__, {ctx, nerves, tty}, opts)
end

def start_link(%Context{} = ctx, tty, opts) when is_binary(tty) do
  GenServer.start_link(__MODULE__, {ctx, tty}, opts)
end
```

Figure 16 : Démarrage du GenServer

Voilà ci-dessus le code de démarrage du GenServer qui va initialiser la communication avec la Raspberry Pi qui possède un module UART.
L'OS génère des Messages en G_codes qui sont analysées en langage Elixir et elles sont envoyées à la Raspberry Pi.

```
10    def parse_code("R00 Q" <> tag), do: {tag, :idle}
11    def parse_code("R01 Q" <> tag), do: {tag, :received}
12    def parse_code("R02 Q" <> tag), do: {tag, :done}
13    def parse_code("R03 Q" <> tag), do: {tag, :error}
14    def parse_code("R04 Q" <> tag), do: {tag, :busy}
15
16    def parse_code("R05" <> _r), do: {nil, :noop} # Dont care about this.
17    def parse_code("R06 " <> r), do: parse_report_calibration(r)
18    def parse_code("R07 " <> _), do: {nil, :noop}
19
20    def parse_code("R20 Q" <> tag),   do: {tag, :report_params_complete}
21    def parse_code("R21 " <> params), do: parse_pvq(params, :report_parameter_value)
22    def parse_code("R23 " <> params), do: parse_report_axis_calibration(params)
23    def parse_code("R31 " <> params), do: parse_pvq(params, :report_status_value)
24    def parse_code("R41 " <> params), do: parse_pvq(params, :report_pin_value)
25    def parse_code("R81 " <> params), do: parse_end_stops(params)
```

Figure 17 : Quelques Gcodes Analysés

Dans la figure 15 la fonction *parse_code* permet d'envoyer une commande en fonction du message G_code qu'on souhaite envoyer au robot. *La méthode parse_code provient du module Farmbot.Serial.Gcode.Parser.*

Nous avons aussi l'analyse des paramètres en Gcode comme la figure suivante nous le montre :

```
def parse_param("25"), do: :movement_enable_endpoints_x
def parse_param("26"), do: :movement_enable_endpoints_y
def parse_param("27"), do: :movement_enable_endpoints_z
```

Figure 18 : Analyse des paramètres Gcode

Ici par exemple la fonction parse_param est une fonction qui permet d'envoyer une commande en fonction du G_code que l'on veut envoyer. La Raspberry Pi comprend alors et traduit les paramètres des G_codes en des commandes pour l'arduino.

Ainsi pour pouvoir rajouter des fonctionnalités, il suffira de modifier ce code qui se situe sur le Github de Farmbot.

3.3 La détection des mauvaises herbes

Le Farmbot dans sa version initiale contient une fonctionnalité permettant la détection des mauvaises herbes. Cela est réalisé via une caméra mise en place sur le Farmbot. L'action est ensuite réalisée grâce à la tête prévue à cette effet. Tout d'abord, la caméra prend une photo. Celle-ci est ensuite récupérée par la raspberry pi qui va utiliser OpenCV pour détecter les plantes sur la photo. OpenCV est une librairie graphique libre qui permet le traitement des images et notamment le calcul d'histogrammes de couleur. Ici on

applique des masques. L'utilisateur définit une couleur comme celle de ces plantes (variante de vert). Un masque de cette couleur est passé sur l'image pour en ressortir les endroits correspondant à cette couleur. Ensuite on place un cercle rouge autour des zones détectées et on enregistre les coordonnées des centres de ces cercles. Grâce à ces coordonnées et aux localisations des plantes déjà plantés, on peut donc déterminé que les plantes restantes sont des mauvaises herbes. On stocke ensuite les coordonnées de ces mauvaises herbes pour les envoyer dans la séquence réalisant le désherbage. Les mauvaises herbes qui sont trop proche des plantes ne peuvent pas être détectées. Il faut donc les retirer à la main.

```
# Remove known plants and their safe zones from the mask
for plant in self.plant_db.plants['known']:
    point = p2c.plant_dict_to_pixel_array(
        plant, extend_radius=self.plant_db.weeder_destrut_r)
    cv2.circle(self.images['morphed'], (int(point[0]), int(point[1])),
               int(point[2]), (0, 0, 0), -1)
# Detect the locations of the remaining plants in the mask
self.find(safe_remove=True)
p2c.p2c(self.plant_db)
if not self.plant_db.plants['remove']:
    self.plant_db.plants['remove'] = []
# The remaining plants (if any) should be weeds that can be
# safely removed, but check again against known plants
self.plant_db.identify(second_pass=True)
```

Figure 19 : Fraction du code de détection des mauvaises herbes

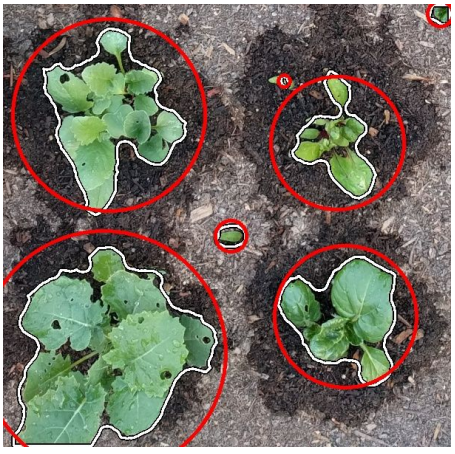


Figure 20 : Détection des mauvaises herbes

4. L'arduino

L'arduino est doté d'un logiciel qui lui permet d'interagir avec tous ces éléments. Le logiciel est responsable de recevoir des G-Code depuis la Raspberry Pi, exécuter le code et renvoyer un résultat. Sur la documentation github, nous avons les explications du fonctionnement de l'arduino dans sa globalité. Il y a différentes listes des codes et des paramètres connus de l'arduino dont voici quelques exemples:

X_ENABLE_PIN	38	X axis enable

Figure 21: Un Paramètre des broches de l'arduino

Ici nous voyons par exemple que le code *X_ENABLE_PIN* permet d'activer en sortir la broche qui permet à l'arduino de contrôler les moteurs qui permettent le déplacement du robot sur l'axe des X.

Code type	Number	Parameters	Function
G			G-Code, the codes working the same as a 3D printer
G	00	X Y Z S	Move to location at given speed for axis (don't have to be a straight line), in absolute coordinates
G	01	X Y Z S	Move to location on a straight line

Figure 22 : Quelques G-codes envoyés à l'arduino

La figure 5 nous montre comment on peut communiquer avec l'arduino avec les G-Codes. Il faut savoir qu'il existe de nombreux G-Codes qui s'écrivent sous la forme [Code Type] + [Number] + [Parameter].
Par exemple un G-Code pour déplacer un bras selon l'axe X s'écrirait : *G 01 X*. Le Code Type est une lettre qui peut varier en fonction de la commande à donner. Ici le G serait une commande de déplacement et le F serait une commande de paramétrage dans notre cas.

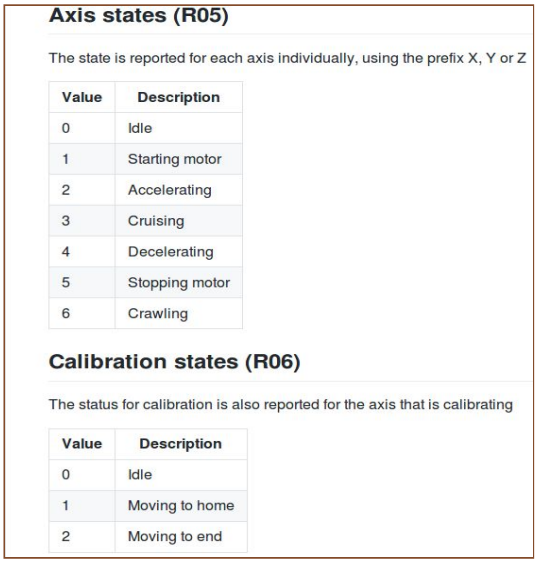
D'autre type de commande sont également utilisés à des fins différentes. Par exemple, les commandes avec le code R sont utilisés pour faire remonter les informations sur la réussite ou non des actions souhaitées ainsi que sur les états des moteurs.

	R	11		X axis homing complete

Figure 23 : Exemple de Code reçu par l'arduino

La figure 5 nous montre comment l'arduino répond aux commandes qu'on lui donne. Ici on est toujours en G-code avec le *TYPE CODE = R* . Par exemple si l'arduino détecte

que le bras a fini son retour à sa position initiale sur l'axe X, il va renvoyer le code *R 11.* Il peut renvoyer aussi des codes concernant les états des moteurs (R05 et R06) et le calibrage des axes. Ces codes sont détaillés ici :



Axis states (R05)

The state is reported for each axis individually, using the prefix X, Y or Z

Value	Description
0	Idle
1	Starting motor
2	Accelerating
3	Cruising
4	Decelerating
5	Stopping motor
6	Crawling

Calibration states (R06)

The status for calibration is also reported for the axis that is calibrating

Value	Description
0	Idle
1	Moving to home
2	Moving to end

Figure 24 : état des moteurs et calibrage

Nous voyons ici que si l'arduino veut renseigner sur l'accélération du moteur, il enverra le code G 05 2. Ou lorsque le bras est en train de revenir à sa position initiale il envoie G 05 1.

Nous avons aussi la liste des paramètres qui peuvent être utilisés dans les commandes (Moteurs, arroseur, Timer...). Voici quelques paramètres que l'arduino connaît :

Parameters	Description	Unit of Measurement
X	X movement	millimeters

Figure 25 : Quelques paramètres connus de l'arduino

Ici nous voyons les différentes lettres qui peuvent être entrés dans les paramètres de G-Codes ([Code Type] [Number] [Parameter]). Ici par exemple nous savons que le paramètre X désigne le mouvement du bras sur l'axe X et le déplacement est mesuré en millimètres.

Enfin nous avons aussi la liste des numéros qui peuvent être entrés dans les G-Codes pour communiquer avec l'arduino sur la figure suivante :

Arduino parameter numbers

ID	Name
2	PARAM_CONFIG_OK
3	PARAM_USE_EEPROM
4	PARAM_E_STOP_ON_MOV_ERR
5	PARAM_MOV_NR_RETRY
11	MOVEMENT_TIMEOUT_X
12	MOVEMENT_TIMEOUT_Y
13	MOVEMENT_TIMEOUT_Z
15	MOVEMENT_KEEP_ACTIVE_X

Figure 26 : Quelques numéros connus par l'arduino

Les différentes actions que fait le Farmbot sont organisés sous forme de séquence. C'est à dire que lorsque l'utilisateur souhaite faire une action, ce n'est pas une seule commande mais bien un groupe de commande qui est envoyé. Ces commandes se font de façon synchrone les unes après les autres.

Prenons par exemple le cas de l'arrosage d'une plante. Les commandes envoyés sont du type :

- 1- Sélection de la tête d'arrosage par la tête universelle
- 2- Mouvement du bras jusqu'au coordonnées de la plante à arroser
- 3- Ecriture sur la PIN correspondant à la gestion de l'arrivé d'eau (ouverture)
- 4- Attendre un temps prédéfinis d'arrosage
- 5- Ecriture sur la PIN correspondant à la gestion de l'arrivé d'eau (fermeture)
- 6- Retour du bras dans la position initiale
- 7- Démontage de la tête d'arrosage

Lorsque nous voudrions ajouter des fonctions pour répondre à notre cahier des charges nous devrions le faire de cette façon.

IV - Nos modifications

1. Objectif 1 : Priorité importante

Dans un premier temps, nous souhaitons réaliser l'ajout de conseils et de suggestions en fonction de la saison. La Web App utilise la base de données OpenFarm, qui est une base de données Open Source. Cette base de données contient des informations tels que l'ensoleillement nécessaire à la plante, les autres plantes à côté desquels elle peut être plantée,l'espacement nécessaire autour de la plante,sa hauteur...Nous allons donc

partir de ce principe pour créer une base de donnée en interne qui indique pour chacune des plantes et des légumes la saison à laquelle il est préférable de planter la graine. Nous pensons la réaliser en local dans un premier temps pour faciliter l'utilisation et les modifications. Pour cela il faut installer la web-app disponible sur le GitHub des constructeurs du farmbot. Nous allons ensuite examiner les plantes déjà présente dans le jardin de l'utilisateur et en fonction de la date, proposer des légumes de saison pas encore plantés. Pour cela nous devons modifier l'interface pour ajouter un onglet "Suggestion". Une modification des codes réalisant l'interface est donc nécessaire (fichier HTML et SCSS). Les plantes déjà plantées sont stockées dans une base de données interne au Farmbot. Nous devrons donc analyser cette table et croiser les informations avec celles de la base de données OpenFarm pour pouvoir faire les bonnes suggestions.

Une cuve de récupération d'eau va être mise en place. Nous placerons un capteur à un certain seuil de la cuve qui sera relié à la Raspberry Pi. Ce capteur se déclenche et affiche une alerte à l'utilisateur pour lui indiquer que le niveau d'eau dans sa cuve de récupération deviendra critique. Nous placerons également d'autres capteurs dans la cuve pour permettre à l'utilisateur de savoir à quel niveau de remplissage est sa cuve. Un affichage de ce niveau sera également mis en place sur la Web App.

A ce jour, les éléments du Farmbot (Raspberry pi, Arduino) sont alimenté en continu. Il n'y a donc pas de gestion d'énergie. Nous souhaitons donc inclure cela. Dans un premier temps, nous pensons alimenter et mettre en service le contrôleur Arduino uniquement lorsque celui-ci est nécessaire pour réaliser une action ou alors récupérer une données. Pour cela, nous allons mettre en place, sur la Raspberry Pi, un programme qui allumera la carte en fonction du type de commande reçu depuis la Web App. Au vu de nos compétences, il nous semble plus facile de réaliser cela en Nodejs. L'étape suivante sera de se concentrer sur la gestion de l'alimentation des différents moteurs. Un choix de la gestion de l'arrivée d'énergie soit via les panneaux solaires ou par l'électricité du secteur est également à mettre en place.

2. Objectif 2 : priorité moyenne

Notre deuxième objectif est d'optimiser la gestion de l'eau. Pour cela, nous allons réaliser une station météo. Nous avions prévu l'achat des différents capteurs pour la réaliser dans nos devis. Les capteurs que nous avons choisi sont : un anémomètre, une sonde de température et d'humidité et un capteur de pression barométrique.

Nous allons ensuite connecter les connecteurs au contrôleur Arduino. Nous allons ajouter des commandes de lecture (R) des broches concernées et ensuite réaliser le processus de transfert des données récoltées jusqu'à la Web-App. Une modification de l'interface est nécessaire pour ajouter un onglet "Meteo". Sur cette onglet se trouve un bouton permettant de récolter les informations de la station météo en temps réel. Un appui sur ce bouton entrainera le processus de lecture des PINs et affichera les données de la station météo.

Nous souhaitons obtenir les prévisions météorologique pour les 7 jours suivant. Pour cela, nous devons récolter les données depuis une base de donnée météorologique

publique. Nous pensons utiliser la base de données OpenWeatherMap. Cette base de données a l'avantage d'être Open Source et de pouvoir nous fournir des prévisions sur 5 ou 16 jours. Il nous faudra juste récupérer la position du Farmbot ou demander à l'utilisateur de nous renseigner le pays et la ville où est positionné le farmbot pour pouvoir faire les requêtes correctement. La réponse à ces requêtes sera au format JSON. Il nous faudra l'interpréter pour ensuite réaliser un affichage correct.


```
{"cod":"200","message":0.0032,
"city":{"id":1851632,"name":"Shuzenji",
"coord":{"lon":138.933334,"lat":34.966671},
"country":"JP"},
"cnt":10,
"list":[{
    "dt":1406080800,
    "temp":{
        "day":297.77,
        "min":293.52,
        "max":297.77,
        "night":293.52,
        "eve":297.77,
        "morn":297.77},
    "pressure":925.04,
    "humidity":76,
    "weather":[{"id":803,"main":"Clouds","description":"broken clouds","icon":"04d"}],}
        ]}
```


Figure 27 : Exemple de document Json renvoyé pour les prévisions météorologique

L'objectif final est de lier les prévisions météo aux ressources que l'on a : nous avons une certaine quantité d'électricité stockée dans la batterie et une quantité d'eau présente dans la cuve. Admettons que la station météo prévoit un temps maussade, et que le niveau des batteries ne permet pas de tenir le temps suffisant, il faut alerter l'utilisateur de ce problème, de même pour la gestion de l'eau. Un programme permettra en fonction de la somme précipitations prévues, du reste d'eau contenu dans la cuve et du besoin en eau pour alimenter le farmbot de déterminer si l'utilisateur devra remplir ou non sa cuve de récupération d'eau. Pour ce qui est de l'ensoleillement, la mesure provisionnelle est plus compliquée à réaliser sachant que les prévisions ne nous donne qu'un avis (nuageux,ensoleillé,pluvieux…). Nous pouvons donc avertir l'utilisateur seulement si un mauvais temps est prévu sur une durée qui pourrait nuire à l'alimentation du Farmbot par les panneaux solaires.

3. Objectif 3 : priorité basse

La mise en place de l'outil de retournement de la terre est la tâche la plus complexe. Dans un premier temps il faut réaliser l'outil à l'aide d'un logiciel de 3D, par exemple SolidWorks nous permettra de réaliser cet objet. Il faudra ensuite utiliser une imprimante 3D (une imprimante est présente au FAB Lab à côté du PNRB et présente également au Bâtiment 6 au campus de Beaulieu.). Pour faire l'objet, nous utiliserons les plans fournis pour les autres têtes. En effet, toutes les têtes doivent avoir la même base pour pouvoir réaliser la connexion avec la tête universelle. Ensuite, il nous faut créer de nouvelles commandes (G,F et R) qui nous permettrons, comme c'est actuellement le cas pour les

autres outils, d'effectuer les actions de retournement de la terre. Une modification de la Web App et notamment de l'interface pour y ajouter un outils dans l'onglet "tools".

Pour utiliser cet outil de retournement de la terre, nous avons pensé procédé en utilisant la séquence suivante :

- 1- Sélection de la tête de retournement de la terre par la tête universelle
- 2- Mouvement du bras jusqu'au coordonnées de la zone où l'on souhaite retourner la terre
- 3- Mouvement du bras d'avant en arrière sur une distance définies (ex 15cm) par l'utilisateur un certain nombre de fois.
- 4- Retour du bras dans la position initiale
- 5- Démontage de la tête de retournement de la terre

Pour ce qui est de la stagnation de l'eau, nous avons réfléchi à plusieurs systèmes différents, nous en avons trouvé deux qui peuvent éventuellement être utilisés ensemble.

Tout d'abord, un moyen naturel permettant d'éviter l'apparition d'algues nocives est de mettre à l'abri la cuve des rayons du soleil. Nous avons choisi une cuve non opaque. Ainsi, l'eau est protégée des rayons mais, il faudrait tout de même la mettre à l'ombre. Il faut aussi rajouter du charbon actif, le charbon est utilisé pour filtrer l'eau du robinet, il peut être utilisé en agriculture. En effet, il permet de minéraliser l'eau et d'équilibrer le pH. Il faut pour cela acheter du charbon, et en mettre un petit sac dans la cuve.

Une autre solution est de fabriquer un système d'aimant couplé à un moteur. On fait tourner l'aimant dans la cuve à l'aide du moteur, ce qui permet de créer un courant permettant d'éviter que l'eau stagne. Cependant il y a des limites à ce système : il faut un gros aimant pour déplacer un volume d'eau aussi important, en effet la capacité de la cuve est 300 Litres. Il y aura d'une part un problème de choix, un couple moteur/aimant de cette taille n'est pas disponible et le prix n'est pas le même que la méthode naturelle.

En conclusion, nous choisissons d'utiliser le charbon pour la gestion de la qualité de l'eau.

V - Nos capacités

Nous pourrions rencontrer des difficultés durant notre projet. C'est pourquoi nous avons trouvé un forum spécialement conçu pour le farmbot et ses utilisateurs. Nous pourrons consulter le forum à l'adresse suivante : https://forum.farmbot.org/.

Une difficulté qui est sans doute la plus importante, est notre manque de connaissance dans les langages utilisés. Nous n'avons aucune notion en Ruby et en Elixir. Par ailleurs, nous avons quelques notions en CSS et HTML et nous apprenons actuellement à coder en Javascript. Il convient donc de se familiariser avec les différents langage.

De plus, une des difficultés est en train de se résoudre avec le montage du Farmbot. En effet, nous avons reçu de l'aide extérieure afin de le monter. La partie montage est en très bonne progression, il faut donc s'occuper de la partie logiciel qui constitue le coeur de notre projet.

La mise en place des panneaux solaires et de la cuve de récupération d'eau est aussi tâche essentielle qu'il nous reste à faire.

Conclusion :

En conclusion, nous pouvons dire que même si notre implication et notre motivation pour ce projet est toujours importante, la charge de travail restante est tout aussi importante. Nous ne pensions pas que la partie logicielle était aussi complexe. La compréhension du fonctionnement du Farmbot est essentielle pour nous permettre d'apporter ensuite nos modifications. Cela nous a pris un temps important à cause notre manque de connaissance dans certains langages de programmation (Elixir, Ruby…). Nous avons donc pris plus de temps que prévu pour cette tâche ce qui explique notre retard au niveau de notre diagramme de GANTT.

Pour les semaines à venir, nous allons accentuer notre implication notamment dans la partie programmation pour rattraper ce retard et mener à bien ce projet.

"J'atteste que ce travail est original, qu'il indique de façon appropriée tous les emprunts, et qu'il fait référence de façon appropriée à chaque source utilisée"

Sources :

Rick Carlino and Rory Aronson. FarmBot, Récupéré sur gitbub.COM: https://github.com/FarmBot
[Accédées depuis 20 oct. 2017]

- https://github.com/FarmBot/farmbot_os
- https://github.com/FarmBot/Farmbot-Web-App
- https://github.com/FarmBot/farmbot-arduino-firmware
- https://github.com/FarmBot/farmbot-js
- https://github.com/FarmBot/farmbot_os/blob/4006ed212edc14ab13b3648cddffa01e7f265b68/lib/farmbot/serial/handler.ex
- https://github.com/FarmBot/farmbot_os/blob/4006ed212edc14ab13b3648cddffa01e7f265b68/lib/farmbot/serial/gcode/parser.ex
- https://github.com/FarmBot/farmbot_os/blob/4006ed212edc14ab13b3648cddffa01e7f265b68/lib/farmbot/serial/gcode/parser.ex

High Level overview du Farmbot, Récupéré sur farmbot.io https://software.farmbot.io/docs
[Accédé depuis le 20 oct. 2017]

Margareth Rouse, Tech Target Récupéré sur techarget.COM: http://searchmicroservices.techtarget.com/definition/Remote-Procedure-Call-RPC
[Accédé depuis le 25 oct. 2017]

Code de module réalisé par l'équipe Farmbot (notamment la détection des mauvaises herbes) :https://github.com/FarmBot-Labs
[Accédé depuis le 25 oct. 2017]

Site de la base de donnée OpenFarm : https://openfarm.cc/
[Accédé depuis le 15 nov. 2017]

Site de la base de donnée météorologique :https://openweathermap.org/
[Accédé depuis le 20 nov. 2017]

Annexes

Annexe 1 : G-Codes pour Arduino

Source :https://github.com/FarmBot/farmbot-arduino-firmware

Code type	Number	Parameters	Function
G			G-Code, the codes working the same as a 3D printer
G	00	X Y Z S	Move to location at given speed for axis (don't have to be a straight line), in absolute coordinates
G	01	X Y Z S	Move to location on a straight line
G	28		Move home all axis
F			Farm commands, commands specially added for the farmbot
F	01	T	Dose amount of water using time in millisecond
F	02	N	Dose amount of water using flow meter that measures pulses
F	09		Reset emergency stop
F	11		Home X axis
F	12		Home Y axis
F	13		Home Z axis
F	14		Calibrate X axis
F	15		Calibrate Y axis
F	16		Calibrate Z axis
F	20		List all parameters and value
F	21	P	Read parameter

F	22	P V	Write parameter
F	23	P V	Update parameter (during calibration)
F	31	P	Read status
F	32	P V	Write status
F	41	P V M	Set a value V on an arduino pin in mode M (digital=0/analog=1)
F	42	P M	Read a value from an arduino pin P in mode M (digital=0/analog=1)
F	43	P M	Set the I/O mode M (input=0/output=1) of a pin P in arduino
F	44	P V W T M	Set the value V on an arduino pin P, wait for time T in milliseconds, set value W on the arduino pin P in mode M (digital=0/analog=1)
F	51	E P V	Set a value on the tool mount with I2C (not implemented)
F	52	E P	Read value from the tool mount with I2C (not implemented)
F	61	P V	Set the servo on the pin P (only pin 4 and 5) to the requested angle V
F	81		Report end stop
F	82		Report current position
F	83		Report software version
F	84	X Y Z	Set axis current position to zero (yes=1/no=0)
E			Emergency stop

Annexe 2 : Paramètres pour la commande

Parameters	Description	Unit of Measurement
X	X movement	millimeters
Y	Y movement	millimeters
Z	Z movement	millimeters
A	X speed	steps/second
B	Y speed	steps/second
C	Z speed	steps/second
Q	Queue number	#
T	Time	seconds
N	Number	#
P	Parameter/pin number	#
V	Value number	#
W	Secondary value	#
E	Element (in tool mount)	#
M	Mode (set pin mode)	0 = output / 1 = input
M	Mode (read/write)	0 = digital / 1 = analog
XA	End stop 1 on x axis	0/1

XB	End stop 2 on x axis	0/1
YA	End stop 1 on y axis	0/1
YB	End stop 2 on y axis	0/1
ZA	End stop 1 on z axis	0/1
ZB	End stop 2 on z axis	0/1