FARMBOT ALICE PROJECT

Tutor: Nicolas Drougard

Intern : Antoine Sfeir
DCAS Department ISAE SUPAERO

***Abstract*—Reaching other planets is one challenge but surviving and settling there permanently is another challenge to be faced. Supplying or sending food from earth could be considered as an impossible task. So, that's why there's a need for a precise and autonomous agriculture. The Farmbot could have a big impact on this matter and could contribute to the solution of the problem. In this work, the goal would be to use the Farmbot to plant Seeds using classical planning. But, also take photos of these plants and use Artificial intelligence to be able to analyse them and modify the planning accordingly in order to cultivate smartely.**

I. Introduction

This project is a part of the ALICE project (AI for Life In spaCE) with the support of the Innovspace (Fablab of ISAE-SUPAERO). The main aim of this paper is to explore autonomous agriculture, which is applied in space, or on earth. The autonomy is needed so that the workload demanded by astronauts in space would become minimal. This paper also focuses on precision in agriculture in order not to use a lot of resources nor a lot energy. Hence why, Artificial intelligence will be used here. To explore these subjects, the Farmbot is used, mainly because it can achieve all the possible requirements discussed in this paper. (a more detailed description of the Farmbot will be found later in this paper).

II. Overall plan of this paper

- In the beginning of the paper there will be definitions. These definitions are essential for the long term project described just now. But, it should be mentioned that some definition were not directly used in the work done in the part described in this paper. But, some essential information were necessary to understand the whole environment of this project. Thus, mentioning them is essential. This paper is a part of the long term project, so it should describe well the majority of the project environment.
- Later in this paper there will be the overall goal and the application on the robot.

III. Definitions

In this section of the paper, there will be a description of the processes and the tools that will be used to explore the challenges in question in this paper.

A. Farmbot

There is a raised planter bed located behind the campus restaurant Sodexo close to Fablab innovspace. A commercial FarmBot is installed over the planter bed frame. This CNC robot may travel to any location in the garden from the soil level to 0.4m above. The FarmBot also features a magnetic universal tool mount (UTM) on its Z-axis that can automatically swap between tools stored on the west side of the bed. The Farmbot has great precision. The tracks and the stepper motors that the system relies on are the source of this. Although some difficulties in the terrain may result in some imprecision. As a result, the robot can function in an open loop due to the fact we have a precise idea of the steps it is taking. The has multiple features:

1) Water Nozzle: To build a functional autonomous garden, there are custom tools for watering and pruning. The watering nozzle creates a wide and uniform output stream, which leads to better soil water retention. The watering nozzle is attached to the UTM in the central bar of the Farmbot.
2) Z-Axis Sensors: A camera is located on the arm of the Farmbot on the Z-axis. It allows for close-up images of plants and soil. There is also a Sharp infrared distance sensor to measure the height of plants. Both integrate with the FarmBot OS. The camera is next to the UTM.
3) Soil Moisture: To measure soil moisture and model soil dynamics. The Soil Moisture Sensor can be attached to the UTM, after it is taken from its bin.
4) Seed injector: The Seed injector can take seeds from the Seed bin and inject them in the Soil. The Seed Injector can be attached to the UTM, after it is taken from its bin.
5) Weeder : It removes the unwanted weeds that need to be removed. The Weeder can be attached to the UTM, after it is taken from its bin.

B. Artificial Intelligence

Artificial Intelligence consists of a system training on models to take decisions and actions. It is very good in detail-oriented missions because once trained the number of errors committed is low. AI is the umbrella term for technologies

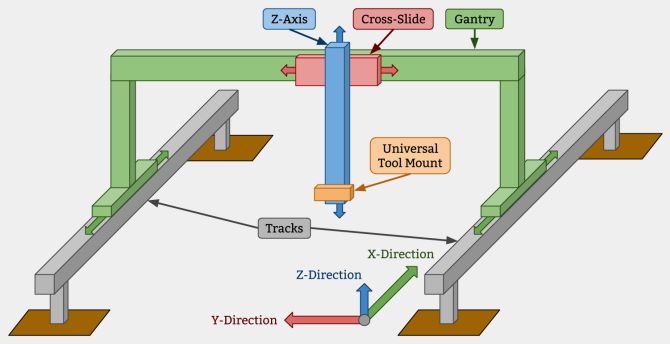

Fig. 1. Farmbot bot axis and components.

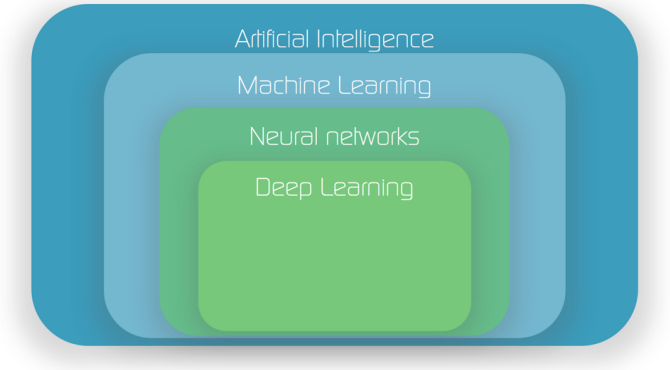

Fig. 2. AI and DL

that solve problems autonomously by simulating human intelligence.

C. Deep Learning

Deep learning is a part of Machine Learning (Machine learning is a subset of AI). It is just that AI is a broader term to include wider and more developed abilities and functions. It consists of imitating how the human brain functions. So, it learns from large amounts of data. In deep learning there are neural networks. Neural networks are a collection of algorithms designed to process data and produce a prediction with the least amount of errors. It processes and analyses a lot of inputs and other variables to produce the output. Using this link Youtube video Link There are many types of neural networks: cnn(convolutional neural network), Long-short memory network and so on. To explain neural network, we have to think of the neuron as something that holds a number. We will refer to a neuron being activated if the number is high and not activated if it is low. There is a layer of neurons. The number of neurons in the layer is variable depending on the problem in hand. The network is composed of many layers of neurons. The idea is that from the first to the last layer progressively through each layer the system through calculations gets closer to the solution and activates some neurons more and some neurons less. So that, at the end the last layer would give the solution. The solution would be in the form of some neurons in that last layer being activated and some not. To do to that, the system has been engineered to use something like a cost function when the machine is training on models. In other terms, the cost function would indicate to the system if it is functioning well or not, if it is giving a good answer or not. Bearing in mind, that while training the system the expected answer is already known. Now the system will try to minimize this cost function using multiple mathematics skills. Various methods are available to minimize the cost function back propagation, forward propagation and gradient descent.

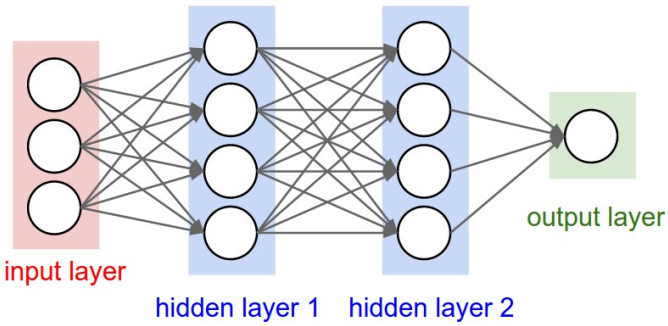

Fig. 3. Normal neural network

D. CNN

To be able to explore our targets, an understanding of CNN (convolution neural network) is required, especially with Image processing. The main idea in CNN is that there are convolutional layers instead of normal layers of neurons. This means the system uses convolutions to transform the input to the output and give it to the following layer. CNN is very suited to pictures because CNN architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. CNN is described in a good way in a Standford Course which can be found online: [1] Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function on the last (fully-connected) layer.

In each convolutional layer there are learnable filters. In general, these filters are small spatially. During the forward pass, we convolve each filter with the whole input volume. The convolution done would be a dot product done between the 2. Convoluting the filter will produce a 2D activation map that represents the response of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or

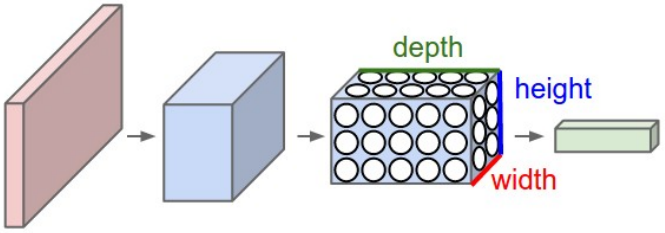

Fig. 4. CNN network

eventually entire honeycomb or wheel-like patterns on higher layers of the network. Then, at the end the filters would be completed and all the activation maps obtained will be stacked.

E. Image Processing

Let's understand how images are perceived by a computer. For example a black and white image is a matrix of nxn pixels where every pixel will indicate the intensity of the black. In the case of colored pictures we have 3 nxn matrixes stacked over each other. So,we have a nxnx3 representation.

In image processing, we have:

- **Image classification :** It is the process that consists of the computer recognizing which type of objects are present in the picture in question.
- **Image Localisation :** It consists of finding where is the location of the object in the picture
- **Object segmentation :** It is the action of finding the perimeter of the object in question.
  To be able to recognize elements in the images, a dataset should be taken and annotated manually. In that way, the system knows what to identify in the images.

Application on the Farmbot: Localisation is not needed for the plants because the location of the plants is already known because the robot planted them in a specific position. But localisation may be needed to eliminate the weeds.

F. Classical planning

The system will function by itself using a methodology called classical planning. In a simplified way, the classical planning wants to find the steps to take to arrive at a certain goal. It is not a closed loop system because it is a prediction of how the system will be and the steps that should be taken in the future to arrive at the goal set in the first place. The language used to represent the classical planning is the PDDL (Planning Domain Definition Language). In that way, 2 files should be done: Domain.pddl and Problem.pddl. In these 2 files there should be defined the predicates, the actions and the goal.

- The predicates in general describe the state of the system. It describes the system in a piece of code. The solver will have an idea of the states of the system through the predicates written in the domain.pddl file.
- The actions are also described in the domain.pddl file. The actions indicate how the system can pass from a state to another. Also, it indicates the conditions required so that a specific action can be done.
- The goal is defined inside the problem.pddl file. It indicates the finish goal, that the system should achieve using the available actions.

In order not to write the 2 pddl files in the pddl language, there is a python library "py2pddl". This library allows to write the python code and the library will write by itself the problem.pddl file and the problem.pddl file.

The next step would be to use an ff solver, that reads the 2 pddl files and finds the optimal solution to arrive to the goal mentioned. There are multiple ff solvers available online.

At the end, the solver will give a detailed plan containing the order of the actions that need to be done to arrive to the goal wanted.

Classical planning is a critical part of AI. Being able to execute the demanded actions and ordering them is an essential part. Also, classical planning is one of the earliest forms of AI.

G. Markov Decision Process — MDP

The MDP consists of 4 entities:

1) S: State
2) A : Action
3) P(S,S') : Probabilities of moving from a state to state
4) R(S) : Reward after transitioning to state S

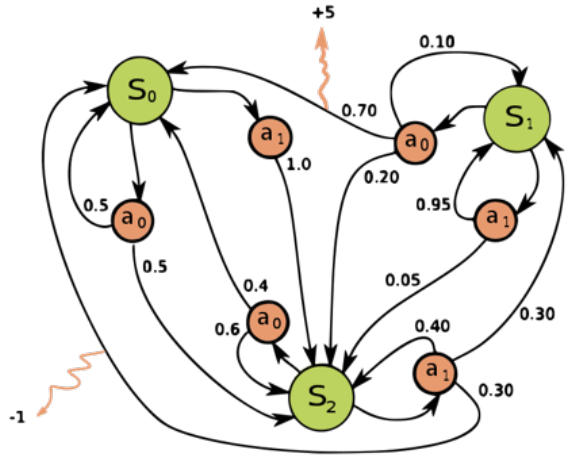

Fig. 5. MDP diagram2 [3]

To explain more:

- States: The system at different times goes from state to state. The system can be in $State_i$ equipped with certain tools. Then, perhaps in another location in $State_j$ equipped maybe with same tools or not.
- Actions : Actions make the system go from $State_i$ to $State_j$.
- P(S,S'): The system in $State_i$ wants to go to a different State $State_j$ there is a specific probability. That's because

in theory the system can go to different states with different probabilities.
- R(S): To make the system priorities some results, the user wants to see. We introduce the reward function which will give a reward for the system, if advancing to a state has benefited our cause. On the other hand the reward function can penalise the system for moving to particular states.

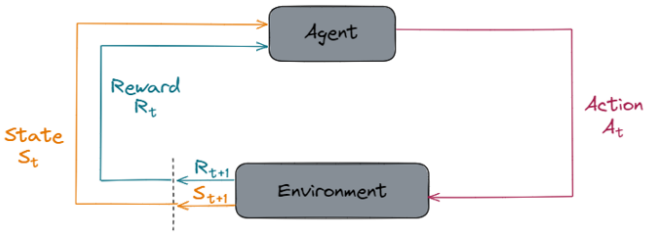

Fig. 6. MDP diagram [4]

In summary, there is the Agent, which does an action. The environment then will be in a different state, but also it will generate a reward coming from the reward function. To do this a transition matrix and a reward function should be defined. Plus, there is a reward matrix and a transition matrix. The reward matrix has 2 dimensions: (Lines : 1, Columns : Number of States). The Transition matrix has 3 dimensions (Number of actions: n, number of states : s, number of states s). In simpler terms, it is a matrix containing n matrixes of size s by s. This transition matrix contains the probabilities of transitioning from a state to another depending on the action selected.

IV. Related Works

To be able to understand and advance in the project, related works were used. And in this section, there is a description of some key learned aspects.

A. Alpha Garden

[2] Alpha Garden is an interesting project using similar tools and similar methodology. The research was performed at the AUTOLAB at UC Berkeley in affiliation with the Berkeley AI Research (BAIR) Lab, and the CITRIS "People and Robots" (CPAR) Initiative. t. Their project, AlphaGarden, is once again a perfect mixture of autonomous cultivation, artificial intelligence, machine learning and agricultural science. Its main purpose is the exploration of Polyculture farming. It is a sustainable way of farming where multiple crops are intermixed. To study this new way, the Farmbot is used as a testbed to prune and irrigate plants. The principal goal, in fact, is providing multiple cultivation strategies that facilitate plant diversity while maintaining high canopy coverage. Some interesting methods were developped.

1) VARPOD seed placement algorithm: The algorithm tries to assure the seed is placed with at least a radius R_i away from other seeds. Also, while placing the seed the algorithm tries to find the best point p to place the seed. It does that using a weighting probability procedure. In short, the algorithm tries to find the best point p suitable for the best conditions of life to the plant. And that's why sometimes the Farmbot in their case will plant less seeds than other times because it will figure out that there was not enough good points to plant.

2) Plant phenotyping: Also, the system was trained with a machine learning algorithm to do the phenotyping of the plant.

The phenotyping that was used in the Alpha Garden project can be explained in a simplified way we have a picture composed of a number n of RGB pixels (n x n pixels). The system will train to transform this picture into a vector n x n x a. a being a number representing the type of plants that could be in the picture. a could also represent an unknown entity which could be the soil. So for each element in the picture *(x,y,i) i* represents the type of plants.

B. Master's thesis Marguerita Pomilio

- The document gives a good description of the Farmbot installed in ISAE Supaero. Also, it describes the physics and the way the Farmbot is assembled.
- Since the automation of the Farmbot is needed, the use of the classical planning is essential.
- The classical planning which will be discussed in the later stages of this paper is radically different from the one applied in the Master's theses. But, the Master's thesis gave the foundation to build the final classical planning that we generated.
- The use of the py2pddl library has been inspired from this document. That's because the original problem.pddl and domain.pddl files were generated using the py2pddl files.
- The domain.pddl files did not take into account that the robot cannot carry more than one tool at a time. Also, it did not take into consideration that the robot should go and bring the tool from the base if it wants to change the tool in hand. Hence, some modifications were required.
- The final goals required remained broadly the same, some minor modifications were required.
- Some different conditions were added for the execution of some actions.

In short, the Master's Thesis gave this internship good foundation to start from. Plus, it helped discovering the real environment and the problems available.

V. The Goal

In fact, the goal that needs to be achieved is to have a closed loop where the Farmbot is the essential part. Then, the whole system will use The data from the Soil Sensor and the Image processing (with the help of the Artificial Intelligence) to know the growth of the plants to be able to do the actions required (watering in the right places and the required amount of water).

VI. Classical Planning on the Farmbot

A. Farmbot in ISAE

To explain the complications of installing the farmbot encountered at ISAE, explanations of how the farmbot works

were given by the Master Thesis by Ms.Pomilio. In fact, there is a webapp installed on farmbot servers. Normally, the user connects through his internet connection to these servers and controls the farmbot in that way. In ISAE the internet connection is private due to high security demanded by the government. And that means that the installation of the webapp on a local server was required. In that way, the farmbot functions without the need for internet, locally with a local server and a local router. In some ways, this is similar to what will happen on Mars, mainly because there is no internet on that planet either. Now this solution leads to some software problems. One example is the camera which stopped working mainly because of different versions of software between the camera and the server.

The Classical Planning: First of all the problem.pddl and the domain.pddl files are written and the corresponding predicates and actions are added. Then these 2 files should be solved using the ff solver. The ff solver used in our project to solve the classical planning is available in the sources showed at the last paragraph of this paper. In return, a clear path of the actions that should be done in the order given will be available. So, now controlling the robot would be the next logical step. To be able to control the Farmbot via sending codes to be executed, an explanation of how the Farmbot functions should be given.

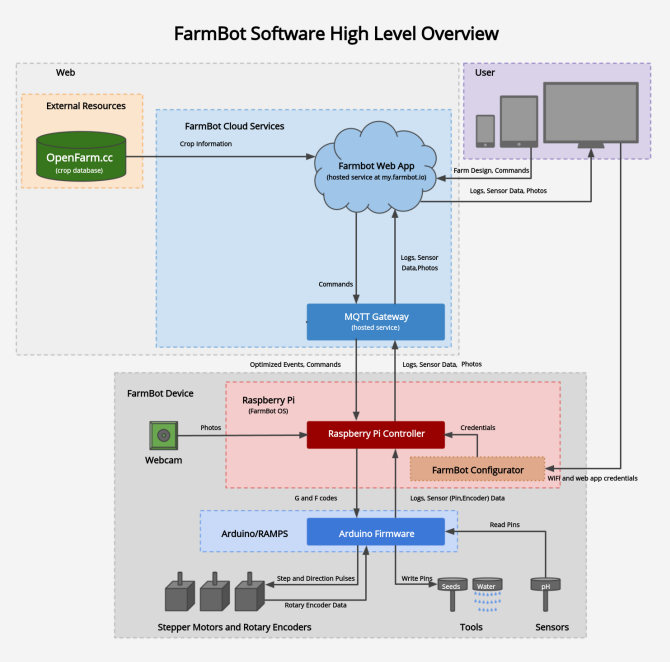

Fig. 7. Farmbot Software high Level Overview

To explain in a simple manner, the Farmbot contains an Arduino-based microcontroller (Farmduino). This microcontroller controls the sensors and the actuators. Using the Arduino as the processing unit for these tasks may result in a poor reception of large memory tasks. To solve this problem and have the possibility to connect to the server and the internet, there is a Raspberry Pi single board computer. This Raspberry Pi is connected to the Arduino. In that way, The Raspberry Pi communicates messages to the end user and the server via the REST API. Also, it maintains the work schedule.

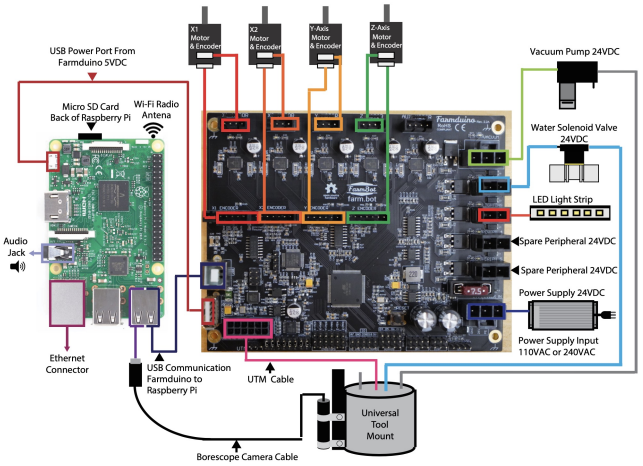

Fig. 8. Farmbot Electric System diagram 1 [5]

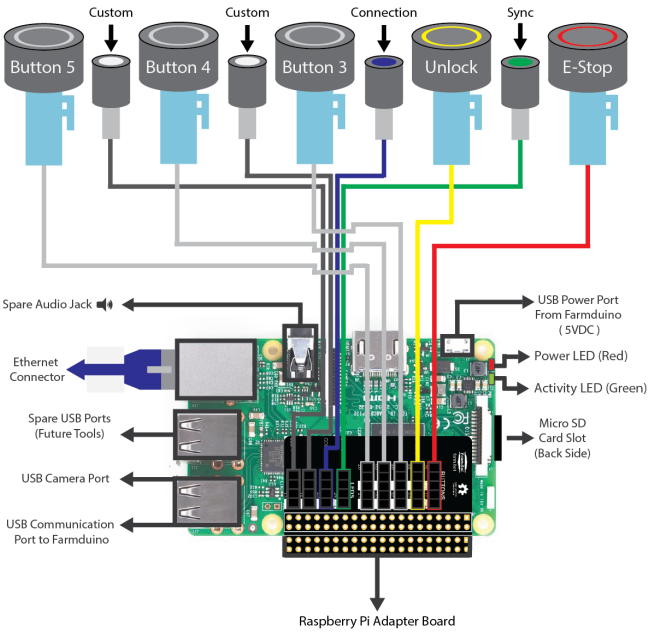

Fig. 9. Farmbot Electric System diagram 2 [5]

As it has been already established, the Raspberry Pi will connect the server to get access to the commands that need to get executed. Normally, there is a WebAPP (or Web Server) containing:

- REST API for data storage and access (The REST API is able to store and contribute to the execution of multiple commands)
- Real time message broker for messaging between the database, user and device
- graphical user interface

To simplify the process, a simple explanation on how to send the information to the WebApp will be given here below:

B. The control of the Farmbot via python

There is an already present python library on github that allows the control of the Farmbot. Here, there will be a description of the way this library has been used in this project. The library can be found using this link Farmbot Python Library

In the example.py file there are functions called on_connect() and on_response(). If all the commands are put in the on_connect() function the commands will be most probably executed in an order different from the one desired. One solution could be using the on_response() function. Each time a command is done the Farmbot enters into the on_response() function with the id of the command that has been executed. In that way, the logic used is to insert the commands one after the other in the on_response() function. It should be mentioned that each command has an id. That's why in the on_response() function an Id check is done and depending on the id of the command that was done previously, the code will choose the next command. In fact, each time a command is executed there is a counter increasing inside the on_response() function. In that way, depending on that counter the actions will be executed one after the other. Here the figure of the on_response() at the early stages below is an example which may help understand more:

```
def on_response(self, bot, response):
    if self.id==response.id:
        print('cnt', self.cnt)
        if self.cnt == 0:
            self.id = bot.move_absolute(x =A[15][0], y=A[15][1], z=A[15][2])
            print("MOVE_ABS REQUEST ID: " + self.id)
        elif self.cnt == 1:
            self.id = bot.take_photo()
            print("TAKE_PHOTO REQUEST ID: " + self.id)
        elif self.cnt == 2:
            self.id = bot.move_absolute(x =A[14][0], y=A[14][1], z=A[14][2])
            print("MOVE_ABS REQUEST ID: " + self.id)
        elif self.cnt == 3:
            self.id = bot.take_photo()
            print("TAKE_PHOTO REQUEST ID: " + self.id)
```

Fig. 10. on_response() at the early stages [5]

The python library contains specific commands that the Farmbot accepts so that it does the corresponding actions. As it has been described above, the classical planning will give us a detailed plan of actions to execute in order to arrive to the desired goal. The thing is the language used to describe the actions generated by the classical planning is not compatible with the language the Farmbot wants to receive. That's why there is a need to create a parser that transforms the result of the classical planning to a language that the Farmbot understands. The parser then stores the result in a txt file which then will be read by the on_response() function to execute the corresponding action, as described in the picture of the on_response() function at the final stages.

The parser reads the commands wanted by the classical planning and matches them with a corresponding piece of code accepted by the Farmbot.

```
def on_response(self, bot, response):
    if self.id==response.id:
        print('cnt', self.cnt)
        program = self.cmds[self.cnt]
        exec(program)
        program = self.cmds[self.cnt+1]
        exec(program)
        self.cnt+=2

    print("ID of successful request: " + response.id)
```

Fig. 11. on_response() at the later stages

C. Storing values of the Soil Sensor

It should be mentioned that each time the Farmbot uses the Soil Sensor it enters in the function on_log(). This feature has been used to store all the data in a csv file. In that way, this data is saved and can be used when needed.

D. The transition to MDP

MDP can simulate improbabilities like for instance if it had rained. Also, it applies the system of reward. In that way, the reward wanted can be applied so that the system evolves the way we want to. To apply the MDP to the Farmbot, some simplifications were introduced to the problem. It should be mentioned that the real system will work using the same logic that the simplified system uses. The simplifications just make the implementation and the explanation easier. The system consists now of a state composed by 3 elements:

1) L : Location of the plant
2) T : Tool chosen (Nothing, Watering Nozzle, Soil Sensor)
3) M : Moisture Level (Unknown, Low, High)

To simplify, further more the system, only 2 plant locations were introduced. With that we have:

1) Locations : $[L_1, L_2]$.
2) Actions : $[Water_1, Water_2, Sense_1, Sense_2, Go_1, Go_2, Unmount, MountWater, MountSoilSensor]$
3) Tools : $[Nothing, WateringNozzle, SoilSensor]$

As a consequence, the state would be: (L_i,T_i,M_i). The order chosen to enumerate all the states should be described. That's because this order gives meaning to the transition and reward matrixes. In that way, the order is: $[(L_1, T_1, M_1), (L_1,T_1,M_2), (L_1,T_1,M_3), (L_1, T_2, M_1), (L_1, T_2, M_2), (L_1, T_2,M_3), ... (L_2,T_1,M_1), (L_2,T_1, M_2), (L_2,T_1, M_3), (L_2,T_2, M_1) ...(L_2,T_3,M_2), (L_2,T_3, M_3)]$.A code has been created to fill the transition and reward matrixes. It is available on github. Each time the system passes from a state where a plant with low humidity to a state where it is watered it is a positive reward for the system.

Some rules and tricks can be followed to implement the transition matrixes.

- If an action does not lead to a change of state there should be a 1 in the transition matrix on the corresponding point.
- P(L_j, T_j, M_j | L_j, T_j, M_j,a) = P(L_j | L_i,a) x P(T_j | T_i,a) x P(M_j | L_i,T_i,M_i,a)
  In that way, instead of writing the transition between all

the states only the matrixes of the "substates" (L_j, T_j, M_j) are written. Not to forget that the matrixes of the sub-states and the dependency on all or some other sub-states will vary from sub-state to another.

- The Reward Function should take into consideration the energy and the time the actions take.
- The Transition Function:
  - For the action $sense_i$:
    - P(M_j = High | M_i = Unknown, $sense_i$) = 0.1
    - P(M_j = Low | M_i = Unknown, $sense_i$) = 0.9
    - P(M_j = Unknown | M_i = Unknown, $sense_i$) = 0
  - For the action $water_i$:
    - P(M_j = High | M_i, $water_i$) = 1
    - P(M_j != High | M_i, $water_i$) = 0
  - For the action go_i :
    - P(L_j = L_i | L_i, go_i) = 1
    - P(L_j != L_i | L_i, go_i) = 0
  - For the action unmount :
    - P(T_j = Nothing | T_i, $unmount$) = 1
    - P(T_j != Nothing | T_i, $unmount$) = 0
  - For the action $MountWater$:
    - P(T_j = Watering Nozzle | T_i, $MountWater$) = 1
    - P(T_j != Watering Nozzle | T_i, $MountWater$) = 0
  - For the action $MountSensor$:
    - P(T_j = Soil Sensor | T_i, $MountSoilSensor$) = 1
    - P(T_j != Soil Sensor | T_i, $MountSoilSensor$) = 0
  - Three "for loops" inside each others were implemented so that the code runs through all the items of the matrixes to fill them with the corresponding values.

So, this is what has been done in the ipynb file coded in the internship. To completely understand what has been coded the order of the states and the order of the actions should be taken into account with a lot of attention.

VII. The overall goal

- As mentioned above, the overall goal is to have a closed loop controlling the garden. The classical planning and the MDP are part of this process. Then, from all the explanation of AI, it should be noted that the image processing algorithms will give a number of pixels. This number gives an idea of the growth of the plant. This should be added as a reward in the MDP designed in this project. Mainly because, there is until now only one reward implemented which is irrigating a plant with low humidity levels.
- Image processing using CNN with the right training will give a clear idea of the growth of the plants in the garden. This, of course, being done with the analysis of the pictures taken regularly of the garden.
- The system can in that way, use the data it has (the analysis of the images and the soil sensor data) to implement the corresponding actions on the garden so that the plants grow in the best way possible. The corresponding actions consist of the watering the plants and using the weeder to eliminate unwanted species.

VIII. Files

All the necessary files can be found on the Gitlab of ISAE SUPAERO corresponding to this project. Also, some files can be on my github: My Github

IX. Conclusion

In conclusion, Classical planning has been applied to the Farmbot in order to achieve particular goals. A parser to translate the language of the ff solver (that generates the result of the classical planning) to a language understandable by the Farmbot has been created. Also, a description of the problem using the Markov Decision Process has been achieved. To continue and achieve the whole big project, the MDP should be solved and optimized. On the other hand, Image processing using CNN should be applied to the Farmbot, so that the system is closed and works towards growing the plants in the best conditions possible.

Antoine SFEIR

References

[1] Ruohan Gao Fei-Fei Li, JiaJun Wu. Cs231n: Deep learning for computer vision. Stanford Course Available at https://cs231n.github.io/convolutional-networks/.

[2] Mark Presten, Yahav Avigal, Mark Theis, Satvik Sharma, Rishi Parikh, Shrey Aeron, Sandeep Mukherjee, Sebastian Oehme, Simeon Adebola, Walter Teitelbaum, Varun Kamat, and Ken Goldberg. Alphagarden: Learning to autonomously tend a polyculture garden, 2021.

[3] author Unknown. Mdp diagram. Figure Available at https://en.wikipedia.org/wiki/Markov_decision_process/.

[4] author Unknown. Mdp diagram. Figure Available at https://deeplizard.com/learn/video/my207WNoeyA/.

[5] author Unknown. Mdp diagram. Figure Available at https://farm.bot//.