

# Bachelor Degree Project



## **CROP AND WEED DETECTION USING IMAGE PROCESSING AND DEEP LEARNING TECHNIQUES**

Bachelor Degree Project in Production Engineering  
G2E, 30 ECTS  
Spring term 2020

Lina Chaaro  
Laura Martínez Antón

Company Supervisors: Peter Ågren  
Fredrik Wingquist

University Supervisor: Rikard Ed  
Examiner: Magnus Holm

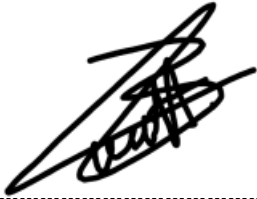
## Abstract

Artificial intelligence, specifically deep learning, is a fast-growing research field today. One of its various applications is object recognition, making use of computer vision. The combination of these two technologies leads to the purpose of this thesis. In this project, a system for the identification of different crops and weeds has been developed as an alternative to the system present on the FarmBot company's robots. This is done by accessing the images through the FarmBot API, using computer vision for image processing, and artificial intelligence for the application of transfer learning to a RCNN that performs the plants identification autonomously. The results obtained show that the system works with an accuracy of 78.10% for the main crop and 53.12% and 44.76% for the two weeds considered. Moreover, the coordinates of the weeds are also given as results. The performance of the resulting system is compared both with similar projects found during research, and with the current version of the FarmBot weed detector. From a technological perspective, this study presents an alternative to traditional weed detectors in agriculture and open the doors to more intelligent and advanced systems.

## Certificate of authenticity

This thesis has been submitted by Lina Chaaro and Laura Martinez Antón to the University of Skövde as a requirement for the degree of Bachelor of Science in Production Engineering.

The undersigned certifies that all the material in this thesis that is not our own has been properly acknowledged using accepted referencing practices and, further, that the thesis includes no material from which I have previously received academic credit.



*Lina Chaaro*



*Laura Martínez Antón*

*Skövde 2020-06-04*

*Institutionen för Ingenjörsvetenskap/School of Engineering Science.*

## Acknowledgements

We would like to start by thanking everyone who has joined us during this thesis, but as we might probably forget someone, we will start by thanking everyone who does not appear in this page so that they do not get upset for forgetting about them.

As we are two different authors, each of us would like to thank her family in her own way:

- Laura: to mum and dad, for always holding me and being there no matter what, even reading this thesis regardless its language. To Alejandro, for being exactly how he is, the best brother I could have wished for; thanks for everything (even helping mum and dad translating this).
- Lina: for my family, thank you for being proud of me and giving me your unconditional support even when not completely understanding most of this project. You are the main reason I have gotten to where I am today, and I will keep you always in mind wherever I go.

To Antón, Iván, Dani and Pablo. You have been a pillar of support at the university, even when it seemed to be never-ending, having you by our side has made it really worth it.

To Anexo, this last year with you has been unforgettable. Thank you for suffering through the thesis all together, even those who did not have to do their own but still suffered with us.

But mostly, to the both of us:

Thank you, Laura, for being my partner in crime all these years, and for all the years to come. Thank you also for the very much needed moral support. This experience would not have been the same without you, and I could have certainly not wished for a better partner.

To Lina, for all the time we have spent together and all the time we still have left. I could thank you for a million reasons, but I would never finish. Thank you for everything in general, both good and bad moments, because those have been moments together and that is what really matters. You are the best partner I could have imagined. And thanks for helping me writing and correcting me, you are great.

## Table of contents

|  |     |
|--|-----|
| Abstract .....                                   | ii  |
| Certificate of authenticity .....                | iii |
| Acknowledgements .....                           | iv  |
| Table of contents.....                           | v   |
| List of Figures.....                             | vii |
| List of Tables .....                             | xi  |
| List of Acronyms and abbreviations .....         | xii |
| 1. Introduction.....                             | 1   |
| 1.1 Background.....                              | 1   |
| 1.2 Problem description .....                    | 2   |
| 1.3 Aim and objectives .....                     | 2   |
| 1.4 Delimitations .....                          | 3   |
| 1.5 Overview.....                                | 4   |
| 2. Sustainability .....                          | 5   |
| 2.1 Environmental sustainability .....           | 5   |
| 2.2 Economic sustainability .....                | 6   |
| 2.3 Social sustainability .....                  | 6   |
| 3. Methodology .....                             | 8   |
| 4. Frame of reference.....                       | 10  |
| 4.1 Robotics and automation .....                | 10  |
| Agricultural robots.....                         | 11  |
| 4.2 Computer vision: image processing .....      | 11  |
| Image processing.....                            | 12  |
| 4.3 Artificial Intelligence: Deep Learning ..... | 15  |

|  |    |
|--|----|
| 5. Literature review .....                               | 17 |
| 5.1 Weed detection .....                                 | 17 |
| 5.2 Image processing .....                               | 18 |
| 5.3 Deep Learning for weed and crop identification ..... | 19 |
| 6. Initial design .....                                  | 21 |
| 6.1 System overview .....                                | 21 |
| 6.2 Prototype .....                                      | 22 |
| 6.2.1 Image acquisition .....                            | 23 |
| 6.2.2 Image classification .....                         | 25 |
| 7. Image processing .....                                | 28 |
| 7.1 Background removal .....                             | 29 |
| 7.2 Histogram equalization .....                         | 30 |
| 7.3 Sharpening .....                                     | 31 |
| 7.4 Glare removal .....                                  | 31 |
| 8. Project development .....                             | 34 |
| 8.1 FarmBot .....  | 35 |
| 8.2 Python .....   | 36 |
| 8.3 Matlab .....   | 37 |
| 9. Results .....   | 44 |
| 10. Conclusion .....                                     | 48 |
| 11. Discussion .....                                     | 49 |
| References .....   | 50 |
| Appendix A: Project prototype programming .....          | 54 |
| Appendix B: Final project code .....                     | 63 |

## List of Figures

|  |    |
|--|----|
| <i>Figure 1: Sustainable development pillars (Kurry, 2011)</i> .....           | 5  |
| <i>Figure 2: Design and creation process</i> .....                             | 8  |
| <i>Figure 3: Imaging process overview (Jähne &amp; Haussecker, 2000)</i> ..... | 12 |
| <i>Figure 4: Pixel representation (Reina Terol, 2019)</i> .....                | 13 |
| <i>Figure 5: Histograms (McAndrew, 2004)</i> .....                             | 13 |
| <i>Figure 6: Filters (McAndrew, 2004)</i> .....                                | 14 |
| <i>Figure 7: Discontinuity detection (Reina Terol, 2019)</i> .....             | 14 |
| <i>Figure 8: Edge detection (Reina Terol, 2019)</i> .....                      | 14 |
| <i>Figure 9: Connected components (Reina Terol, 2019)</i> .....                | 14 |
| <i>Figure 10: Cross correlation (Reina Terol, 2019)</i> .....                  | 14 |
| <i>Figure 11: Structure of an ANN</i> .....                                    | 15 |
| <i>Figure 12: System overview</i> .....  | 21 |
| <i>Figure 13: Sample of FarmBot code</i> .....                                 | 23 |
| <i>Figure 14: Downloading images code</i> .....                                | 24 |
| <i>Figure 15: Downloaded and stored images</i> .....                           | 24 |
| <i>Figure 16: Network training</i> .....                                       | 25 |
| <i>Figure 17: Training progress</i> .....                                      | 26 |
| <i>Figure 18: Training results</i> .....                                       | 26 |
| <i>Figure 19: Classification and evaluation of the network</i> .....           | 27 |
| <i>Figure 20: Confusion chart</i> .....  | 27 |
| <i>Figure 21: Picture taken with FarmBot Genesis camera v1.3</i> .....         | 28 |
| <i>Figure 22: Picture taken with FarmBot Genesis camera v1.5</i> .....         | 28 |
| <i>Figure 23: Picture taken with C925e Logitech webcam</i> .....               | 28 |
| <i>Figure 24: Picture taken with Xiaomi Redmi Note 8 Pro camera</i> .....      | 28 |

|   |    |
|---|----|
| <i>Figure 25: Background removal code .....</i>                     | 29 |
| <i>Figure 26: Background removal .....</i>                          | 30 |
| <i>Figure 27: Matlab's command for histogram equalization .....</i> | 30 |
| <i>Figure 28: Histogram equalization .....</i>                      | 30 |
| <i>Figure 29: Image sharpening in Matlab .....</i>                  | 31 |
| <i>Figure 30: Sharpening.....</i>                                   | 31 |
| <i>Figure 31: Glare removal code .....</i>                          | 32 |
| <i>Figure 32: Glare removal.....</i>                                | 33 |
| <i>Figure 33: Final processing .....</i>                            | 33 |
| <i>Figure 34: Crop and weeds chosen for this project .....</i>      | 34 |
| <i>Figure 35: FarmBot's field dimensions .....</i>                  | 35 |
| <i>Figure 36: FarmBot's sequence diagram .....</i>                  | 35 |
| <i>Figure 37: Image download in Python.....</i>                     | 36 |
| <i>Figure 38: Deleting Images with Python .....</i>                 | 36 |
| <i>Figure 39: Image processing.....</i>                             | 37 |
| <i>Figure 40: Splitting the images datastore .....</i>              | 38 |
| <i>Figure 41: Network modification.....</i>                         | 38 |
| <i>Figure 42: Training options.....</i>                             | 39 |
| <i>Figure 43: RCNN training command.....</i>                        | 39 |
| <i>Figure 44: Network training .....</i>                            | 40 |
| <i>Figure 45: Training progress.....</i>                            | 40 |
| <i>Figure 46: Object detection command .....</i>                    | 41 |
| <i>Figure 47: Labelled image .....</i>                              | 41 |
| <i>Figure 48: Test evaluation .....</i>                             | 41 |
| <i>Figure 49: Detection command in Matlab.....</i>                  | 42 |
| <i>Figure 50: Coordinates obtention in Matlab .....</i>             | 43 |



|  |    |
|--|----|
| <i>Figure 51: Detection accuracy .....</i>                           | 44 |
| <i>Figure 52: Visualization examples .....</i>                       | 45 |
| <i>Figure 53: Coordinate table .....</i>                             | 45 |
| <i>Figure 54: FarmBot UI .....</i>                                   | 54 |
| <i>Figure 55: Move to origin .....</i>                               | 54 |
| <i>Figure 56: Take pictures of a complete row .....</i>              | 55 |
| <i>Figure 57: Move row sequence.....</i>                             | 55 |
| <i>Figure 58: 'Move Y and take picture' sequence .....</i>           | 56 |
| <i>Figure 59: 'Move X' sequence .....</i>                            | 56 |
| <i>Figure 60: Final commands .....</i>                               | 57 |
| <i>Figure 61: Data acquisition using API tokens .....</i>            | 58 |
| <i>Figure 62: Libraries imported .....</i>                           | 58 |
| <i>Figure 63: Token generation code .....</i>                        | 59 |
| <i>Figure 64: Accessing the images .....</i>                         | 59 |
| <i>Figure 65: Path determination .....</i>                           | 59 |
| <i>Figure 66: Image download.....</i>                                | 60 |
| <i>Figure 67: Datastore creation.....</i>                            | 60 |
| <i>Figure 68: Splitting and augmenting the image datastore .....</i> | 61 |
| <i>Figure 69: Network modification and training options .....</i>    | 61 |
| <i>Figure 70: Evaluation.....</i>                                    | 62 |
| <i>Figure 71: FarmBot's picture division .....</i>                   | 63 |
| <i>Figure 72: Final main sequence (I).....</i>                       | 63 |
| <i>Figure 73: Final 'Move row' sequence .....</i>                    | 64 |
| <i>Figure 74: Final 'Move Y and take picture' command .....</i>      | 64 |
| <i>Figure 75: Final 'Move x' command.....</i>                        | 65 |
| <i>Figure 76: Final main sequence (II).....</i>                      | 65 |

|  |    |
|--|----|
| <i>Figure 77: Python image acquisition .....</i>                           | 66 |
| <i>Figure 78: Execute Python command in Matlab .....</i>                   | 67 |
| <i>Figure 79: Loading images to a datastore.....</i>                       | 67 |
| <i>Figure 80: Image processing.....</i>                                    | 68 |
| <i>Figure 81: ImageLabeler app.....</i>                                    | 68 |
| <i>Figure 82: Formatting ground truth table.....</i>                       | 69 |
| <i>Figure 83: Network modification.....</i>                                | 69 |
| <i>Figure 84: Training options and network training.....</i>               | 70 |
| <i>Figure 85: Commands to plot the training progress.....</i>              | 70 |
| <i>Figure 86: Evaluation of the test set of images code .....</i>          | 71 |
| <i>Figure 87: Network training (I).....</i>                                | 71 |
| <i>Figure 88: Network training (II) .....</i>                              | 72 |
| <i>Figure 89: Plant detection code overview .....</i>                      | 73 |
| <i>Figure 90: Code for object detection and coordinate obtention. ....</i> | 74 |
| <i>Figure 91: Function Coordinates.....</i>                                | 74 |
| <i>Figure 92: Obtention of bounding box center .....</i>                   | 75 |
| <i>Figure 93: Final coordinates according to FarmBot .....</i>             | 76 |
| <i>Figure 94: Prediction evaluation.....</i>                               | 76 |

## List of Tables

|  |    |
|--|----|
| <i>Table 1: CNN comparison (Moazzam, et al., 2019)</i> ..... | 19 |
| <i>Table 2: Cameras comparison</i> .....                     | 28 |
| <i>Table 3: Test accuracy</i> .....                          | 41 |
| <i>Table 4: FarmBot weed detector comparison</i> .....       | 46 |
| <i>Table 5: Cordova-Cruzzaty network comparison</i> .....    | 47 |

## List of Acronyms and abbreviations

|      |  |
|------|--|
| Adam | Adaptive Moment Estimation                 |
| ANN  | Artificial Neural Network                  |
| API  | Application Programming Interface          |
| CNN  | Convolutional Neural Network               |
| HOG  | Histograms of Oriented Gradients           |
| HSV  | Hue, Saturation, Value                     |
| IoT  | Internet of Things                         |
| IT   | Information Technology                     |
| RCNN | Regions with Convolutional Neural Networks |
| RGB  | Red, Green, Blue                           |
| UI   | User Interface                             |

## 1. Introduction

One of the newest and most researched technologies nowadays is deep learning. Deep learning is a technique used to create intelligent systems as similar as possible to human brains. It has made a big impact in all types of domains such as video, audio and image processing (Wason, 2018; Sharma, 2019). On the other hand, agriculture is humanity's oldest and most essential activity for survival. The growth of population during the last years has led to a higher demand of agricultural products. To meet this demand without draining the environmental resources the agriculture uses, automation is being introduced into this field (Mehta, 2016).

The present project aims to merge both concepts by achieving autonomous weed recognition in agriculture; this goal will be reached by using new technologies such as Matlab, FarmBot and Python programming, image processing, deep learning and Artificial Neural Networks (ANNs). These concepts will be explained in more detail throughout this document. This thesis will be developed for Naturbruksförvaltningen, a farming school in Töreboda, Sweden.

### 1.1 Background

Robotics and automation have become an emerging subject nowadays; substituting and aiding humans in manual tasks that can become not only tedious and repetitive, but also difficult due to different factors such as precision. In order to go in depth on this technology deep learning has been implemented with the purpose of giving these systems intelligence, making them capable of learning. Examples can be found everywhere, from industries to humankind's daily life.

One of these examples is agriculture, where automation has found solution to some of the challenges faced by farmers on a daily basis such as crop diseases infestations, pesticide control, weed management, lack of irrigation and drainage facilities and lack of storage management (Jha, et al., 2019). As a way to bring this new technology to urban orchards, FarmBot Inc. was created. It is a local startup that is working within advanced precision agriculture through automation and open source technology (Brown, et al., 2017). FarmBot Inc. has developed a series of robots, called FarmBots, to take care of these orchards in an autonomous way while respecting the environment.

Naturbruksförvaltningen Sötåsen aims to teach its students how to combine agriculture and technology. To do so, they intend to introduce a FarmBot into their studies and go a step further, not only programming it to do the basic agricultural tasks, but also by including deep learning to make the system capable of differencing on its own whether there are weeds on the orchard or not.

## 1.2 Problem description

These last years the combination of automation and computer vision has been introduced into agriculture to reduce human workload. The FarmBot used in this project is one example of that combination. Its functions range from the automation of basic agricultural activities such as watering or seeding, to more advanced and complex tasks such as differencing between crops and weeds. This weed detection system is the focus of this project. It is programmed to take pictures of the crop and process them by a manually activated weed-detection software application from FarmBot where the processing is done based on the colours and location of the elements of the picture. This weed detector is the starting point of this thesis.

Why does the weed detector have to be improved? Even if this system seems to be failproof, it is not. There are three main issues that can be considered: firstly, having to manually activate the weed detector application does not reduce the amount of human labour as much as intended. Secondly, basing the detection on colours is not accurate due to the possibility of a change of lighting or the similarity of colours between weed and plants, among other things. Finally, basing the existence of a weed on the location where the FarmBot has previously planted a seed, does not consider a situation where the FarmBot does not necessarily know where all the seeds are located. As a way to solve these issues, this thesis will implement a weed detector software based on deep learning which will be explained in *Section 1.3*.

## 1.3 Aim and objectives

The aim of this project is to implement a different type of weed detection system than the explained in *Section 1.2*, one that makes use of an ANN to differentiate between crop and weed. In order to achieve this, some objectives need to be set:

1. Image capture using FarmBot
2. Image pre-processing with Matlab
3. ANN training using Matlab
4. ANN testing
5. Use the previous pictures to return weed coordinates
6. Compare ANN performance between the one used by FarmBot and the one used in this project

The Matlab system will implement the pre-processing of images and the training of an ANN in order to have a system able to learn from the already processed images and do that processing autonomously. To take those pictures, the FarmBot will be programmed to capture them every certain

time and forward those images so that Matlab can retrieve them. The performance evaluation of the different ANNs will be done in order to determine which one is better, if the one originally used by the robot or the one developed in this thesis.

To achieve these objectives, the robot will be programmed using the User Interface (UI) of FarmBot and the ANN will be trained in Matlab using images retrieved with Python through the FarmBot REST API (Application Programming Interface). The techniques to be used will be computer vision to work with the camera, image processing and deep learning for pattern recognition and ANN training.

The development of the project will be done considering the following points:

- Research on the FarmBot programming environment and weed detector
- Research on computer vision and image processing
- Research on deep learning and its implementation in Matlab
- Develop the code to take the pictures on FarmBot
- Develop the Matlab code with the ANN trained
- Evaluation and comparison of the ANNs

To summarize, the project will be considered complete when the neural network achieves an accuracy of more than the 50% when identifying both crop and weeds in an image captured by the FarmBot. Those crops and weeds used for the project are spinach, dandelions and cleavers.

## 1.4 Delimitations

To establish the boundaries during the development of the project, some factors will be taken into account. The FarmBot programming will not go further than the basic programme needed to take pictures wherever it is necessary. The ANN to be used will be a previously created network available on MathWorks community, it will not be created from the beginning in this thesis, only the last layers will be modified, and then the network will be trained. This thesis will be developed to differentiate between crop and weed with Matlab in order to evaluate the differences between Matlab's ANN and Raspberry Pi's ANN. The crop and weeds used in this project will be grown in Sötåsen, being the crop specifically spinach and the weeds dandelions and cleavers.

Some hardware limitations can be found on the development of the project, such as the camera resolution or the computer characteristics. The camera resolution will limit the quality of the image, which interferes with the final result of the image processing stage. The computer characteristics will affect the ANN training speed. These characteristics will determine the accuracy of the project, so the better they are, the more precise the results will be.

## 1.5 Overview

This thesis is structured in 11 chapters. *Chapter 1* is the introduction to the project and its background. In *Chapter 2*, the sustainability of the project can be found. The methodology followed is explained in *Chapter 3*. *Chapters 4* and *5* are Frame of reference and Literature review, where the theoretical framework and some similar implementations are analysed. From *Chapter 6* to *Chapter 8* the development of the project is explained, *Chapter 6* corresponds to the initial design; in *Chapter 7* a research on different image processing techniques is found, and, finally, in *Chapter 8* the final development of the project is done. After this, *Chapter 9* shows the results obtained through the thesis, which will be later compared with other technologies in *Chapter 10* as the conclusion of the project. Finally, in *Chapter 11* a discussion about the obtained results and their implication in future projects is done. In addition to this, two appendices are included with a detailed explanation of the developed code in each phase of this thesis.



## 2. Sustainability

Nowadays, humanity's lifestyle is using up their resources leading to a lack of them in the future. Sustainable development is the solution to this problem, controlling the usage of actual resources without compromising the future generation needs (Brundtland, 1987). In order to achieve sustainable development, as said by Samaan (2020), it is crucial to balance its three pillars: economic growth, social inclusion and environmental protection, as it is shown in *Figure 1*. This balance tries to be achieved by accomplishing the 17 goals set by the UN.



**Figure 1:** Sustainable development pillars (Kurry, 2011)

As this thesis focuses mainly on the development of a software that autonomously detects both weeds and crops, the sustainability will depend on how it is implemented in real life. Nowadays, this implementation is mainly done with the help of automation. In the case of this study, as mentioned in *Chapter 1*, our automaton is a FarmBot.

### 2.1 Environmental sustainability

From an environmental point of view, sustainable development aims to balance the actual consumption of resources and their production rate. When it comes to the usage of energy and CO<sub>2</sub> emissions, Nouzil, et al., (2017) states that automation in industry is not environmentally sustainable; the amount of energy used needs to be reduced and so do its emissions, which are approximately the 24% of the total CO<sub>2</sub> emitted. On the other hand, a positive aspect of automation is the waste management; reducing waste by dealing with recycling in industry, including agriculture. Another important aspect is the reduction of chemicals used, as the precision of a machine surpasses a human worker, the usage of pesticides is reduced since it will only be used exactly where needed.

As mentioned in *Chapter 1*, this thesis makes use of a FarmBot, its use is not only helpful for waste reduction, using resources such as water and fertilizers in a more efficient way and pesticides only if needed, but also with CO<sub>2</sub> emissions. According to estimations done by FarmBot Inc., (2018) the CO<sub>2</sub> emitted to produce a FarmBot is between 100 and 150kg, and the yearly CO<sub>2</sub> emissions caused by its use is only of approximately 60kg. Furthermore, the existing possibility of powering the FarmBot with solar energy also reduces the CO<sub>2</sub> emissions (FarmBot Inc., 2018).

## 2.2 Economic sustainability

Economically speaking, sustainability refers to the most optimal use of the existent resources in order to attain economic growth. Automation has increased gains for the industry by increasing productivity and accuracy. The cost of this technology has been reduced since its beginning, but it is still a high investment for small entrepreneurs. Nevertheless, economic sustainability is guaranteed.

The economic sustainability of this thesis related to FarmBot can be based on its return on investment. FarmBot Inc., (2018) estimates a period of three years for its products, comparing the costs of producing FarmBot grown products against the costs of store-bought products.

## 2.3 Social sustainability

Social sustainability cares for health and wellbeing of humans. Nouzil, et al., (2017) talks about how the introduction of automation in people's daily life aims to improve human quality of life and reduce safety risk by replacing manual work in repetitive and dangerous tasks by autonomous systems. Even though it seems like this technology only brings benefits, nowadays it is one of the most discussed dilemmas since many may say it only causes unemployment and others that it creates new jobs opportunities.

FarmBot also improves quality of life, by reducing the amount of necessary human supervision needed on orchards. With this technology no jobs are substituted since it has been developed for personal use instead of industrial. In this thesis, besides FarmBot, social sustainability also focuses on the reduction of time a person needs to control the weeds; this job will be simplified by the use of a computer only needing human intervention to take out those detected weeds. In a larger scale if weed elimination was totally automated this would suppose some job loss, but the workers could be relocated by taking care of the maintenance and supervision of the robots used to do that elimination.

In addition to the three sustainable development pillars, this thesis also tries to achieve some of the 17 sustainable development goals. The ones that are most related and relevant are explained below:

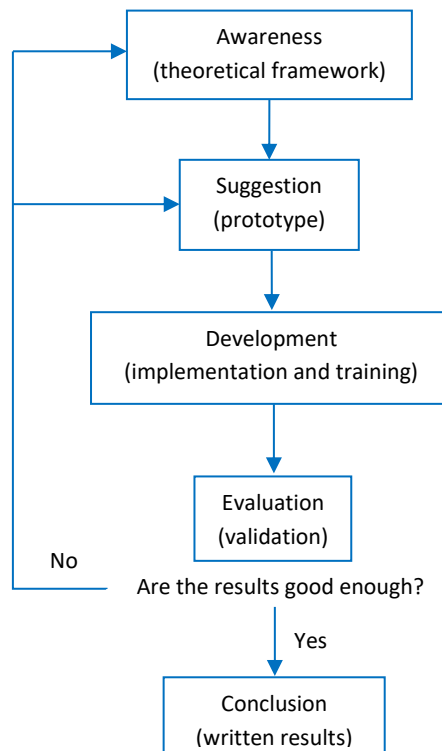
- Quality education since the FarmBot will later be used for educating farming students
- Industry, innovation and infrastructure introducing the new technology of artificial intelligence into backyard robots
- Responsible consumption and production as the FarmBot is used to grow food for the personal consumption reducing overconsumption

### 3. Methodology

As the aim of this project is implementing a weed detection system, it will be reached by creating a program able to identify crops and weeds using image processing techniques and deep learning. This project is not only technical, developing and implementing a program to differentiate crops and weeds with the available technologies; but it is also a research project, since it investigates the already existing knowledge and implementations related to this field of study. As Oates, (2006) says, this type of project contributes to the creation of knowledge, in this case, by introducing a new technique into the functions of FarmBot. To develop this kind of project, the methodology explained in this chapter is followed.

This project's aim is reached by using deep learning to develop a program capable of identifying crops and weeds, therefore the strategy followed will be the one denominated as 'Design and Creation' (Oates, 2006). This strategy focuses on developing new Information Technology (IT) products such as models, methodologies, concepts or even systems to achieve a contribution to knowledge. In terms of this project, this contribution will be the development of a code from which a computer will autonomously identify crops and weeds from images taken by a FarmBot.

Following the Design and Creation strategy involves using an iterative process involving the following steps (*Figure 2*), keeping in mind that each step must be finished before moving on to the next one (Oates, 2006):



**Figure 2:** Design and creation process

- Awareness: recognition of the problem. In the current project, this step englobes the realization of the Frame of Reference and the Literature Review in order to have a deeper understanding of deep learning and choosing the most appropriate image processing techniques. This is developed by doing a research about this project's main concepts, which are robotics and automation, image processing and deep learning.
- Suggestion: design and development of an idea of how the problem could be solved. This will be addressed as the development of a prototype of the program by considering the techniques chosen in the previous step. The prototype will be developed in Matlab, with the functions of downloading the images taken by the FarmBot, pre-processing those images and, finally, training and testing the network.
- Development: when the suggestion is proved to be feasible, it is implemented depending on what kind of IT application it is. This step will lead to the final solution, and it will be divided in two parts in this project: the final programming, which will be based on the previously done prototype, and the training of the ANN to identify and classify both crops and weeds. In order to be implemented, the project will be connected to the FarmBot through Python to download the pictures and then process them in Matlab.
- Evaluation: the application is examined and validated. If the solution obtained does not fulfil the expectations, it is needed to go back to Awareness or Suggestion stage. This step will be done by testing whether the computer is able to identify and classify correctly the crops and weeds. In case it is not, the prototype done in the suggestion step would need to be modified either by changing the pre-processing techniques or the training options given to the network while programming.
- Conclusion: results and the acquired knowledge are consolidated and written. This project focuses the most on the networks performance results such as the accuracy and loss as well as the characteristics of the computer. A comparison will be made with these results and the FarmBot performance.

## 4. Frame of reference

The main concepts of this project are robotics and automation, computer vision and artificial intelligence. For these last two concepts this thesis will focus on image processing and deep learning. Hereafter, the theory related to them is explained in detail.

### 4.1 Robotics and automation

Automation can be defined as “the creation and application of technology to monitor and control the production and delivery of products and services” (Gupta & Gupta, 2015). As it is seen, automation is a versatile technique. Nowadays, it can be found in fields as different as manufacturing, transportation, utilities, defense, security, and many others. The tasks in which it is involved can vary from installation and integration of the technology to the marketing and sales of the industry.

Automation and robotics are closely related concepts. Robotics is a science that covers a broad spectrum of knowledge in order to study and develop systems capable to perform high level tasks, even resembling as much as possible the performance that a human could have or improving it (Angeles, 2014). Those systems are denominated robots, but what is exactly a robot? Mataric (2007) defines robot as “an autonomous system which exists in the physical world, can sense its environment, and can act on it to achieve some goals”. Being autonomous means not being directly controlled by a human, therefore a robot can act on its own as long as it has been programmed to take such decisions. In order to base its decisions on what is around it, the robot needs to be able to sense its environment; this is done by using sensors, devices capable of capturing disturbances of the environment. The information captured is used by the robot in order to act on it, following the steps programmed to achieve a pre-established goal.

Nowadays, both robotics and automation are fast-growing fields and are used in a wide range of applications. Some benefits they have brought to the different industries and sectors are increases in accuracy, precision and efficiency, productivity increased due to a faster task execution, quality improvement and others (Niku, 2020).

There are multiple types of applications where a robot can be used, most of them, as said before, high level applications resembling human performance. One of those uses is agriculture. Robots introduced to agriculture can vary their functionality from large fields maintenance to either small backyards. One of the companies developing these robots is FarmBot, creating systems to carry out the tasks needed to take care of backyard fields

## Agricultural robots

Precision agriculture has led to a significant reduction of the area needed from the whole field down to a sub field level. This scale-reduction could focus even on individual plant care. The more detailed the focus is, the more data is processed; this entails that at a certain level of data, human intervention by its own is not enough to handle that excessive amount of information. Therefore, automation is necessary (Pedersen, et al., 2006; Pantazi, et al., 2020).

Agricultural Robots, also known as Agribots, are robots specifically designed for agricultural purposes. The usage of these robots has been implemented in activities such as seeding, harvesting, weed control, grove supervision, chemicals, etc.; these improve food quality and productivity, labor costs and time, environmental impacts due to agronomic activities, thus reducing pollution and the excess of fertilizers and chemicals. These together lead to an environmentally friendly agriculture.

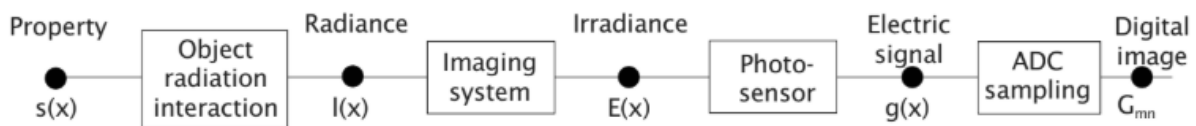
### 4.2 Computer vision: image processing

Computer vision, also known as machine vision, is an area that comprises the acquisition, analysis and processing of images in a way as similar as possible to human vision, using at least an optical imager and a computer to obtain appropriate information (Lu & Lu, 2016; SAS, 2019).

A machine vision system consists of the following elements (Jähne & Haussecker, 2000; Nixon & Aguado, 2019):

- Radiation source: a proper illumination source is needed in order to observe the objects as precisely as possible
- Camera: used to collect the radiation emitted or reflected by the object
- Sensors: they transform the radiation caught into a suitable signal for its future processing
- Processing unit and memory system: to extract features, measure properties and categorize them, as well as to collect and store information about the scene
- Actuators: to react to the final result of the observation

The imaging process (*Figure 3*) involves all the steps of the development of an image from a physical object. Therefore, the imaging system is composed of sensors that transform the radiation into electrical signals which are then sampled and digitalized. The objective of this process is to obtain a signal from a real-life object from which we can determine its properties and geometry through further processing, recognition and classification of objects.



**Figure 3:** Imaging process overview (Jähne & Haussecker, 2000).

## Image processing

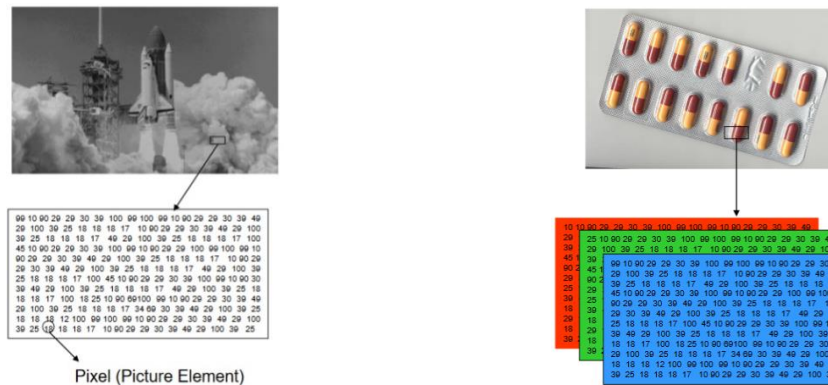
Image processing is a method to perform operations on an image in order to improve it for further analysis. In terms of computer vision, it is called ‘digital image processing’ due to the needed of a digital image to be processed by the computer (McAndrew, 2004).

The most common way image processing is performed into a computer vision system is the following:

1. Image acquisition: the camera and sensors take an image and digitalize it in order to process it.
2. Pre-processing: performing some basic processing tasks in order to have a suitable image to work with.
3. Processing: at this point, all the techniques required for the correct modification of the image are applied.
4. Representation and description: extracting the most particular features of the objects from the already processed image, in order to differentiate these objects.
5. Recognition and interpretation: assigning labels to those objects to completely define them.

As previously mentioned, a digital image is needed in order to be processed. This digital image is understood as a mathematical matrix where every element or ‘pixel’ has its own characteristics depending on the color, brightness, resolution, etc.; the combination of all the pixels organized in a certain way will result in a real-world representation (Figure 4) this is usually represented in two ways: HSV (hue, saturation, value) and RGB (red, green, blue).





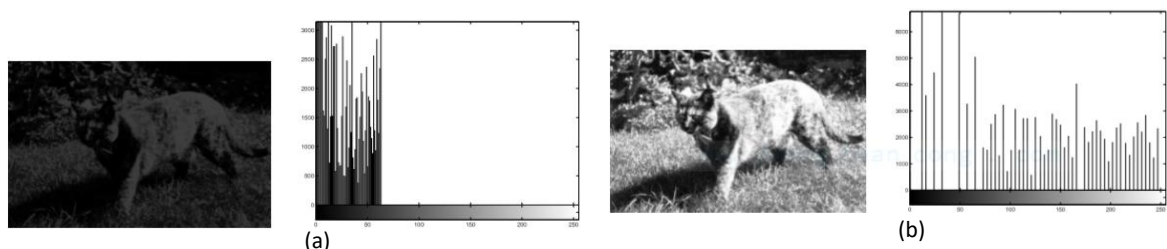
**Figure 4:** Pixel representation (Reina Terol, 2019)

By processing each pixel and modifying its characteristics using image processing algorithms, the image can be improved or even changed completely. McAndrew (2004) divides those different algorithms depending on the tasks they do:

- Image segmentation: divides an image into subimages in order to isolate or identify certain shapes.
- Image enhancement: process an image to make it more suitable for an application.
- Image restoration: reverse the damage done to an image due to a known cause.

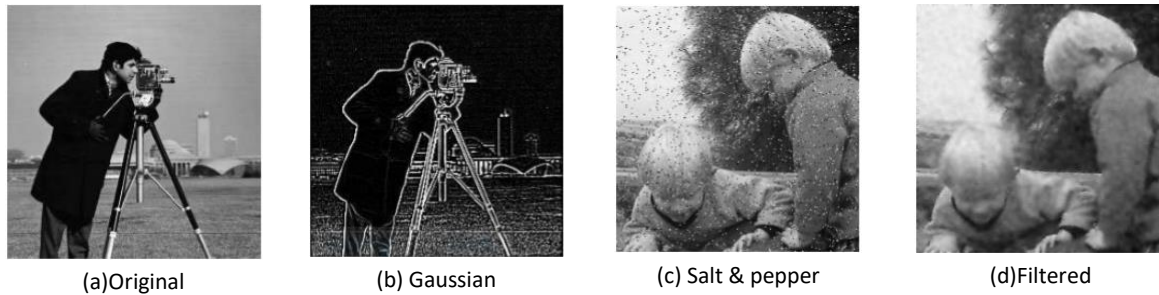
There are lots of different algorithms or techniques that can be used to process an image, hereafter the most used techniques are going to be explained (Reina Terol, 2019):

- Histogram: it shows how many times does a grey level appears in an image (Figure 5). Thanks to this, it is possible to know if the image is too light, too dark, etc.



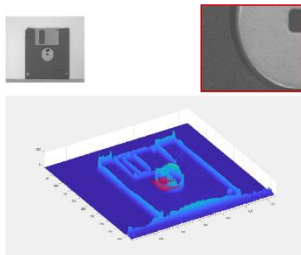
**Figure 5:** Histograms (McAndrew, 2004)

- Filtering: compares a pixel grey level to its neighbors' ones, normally to eliminate noise. There are lots of different types of noises and so are filters (Figure 6).

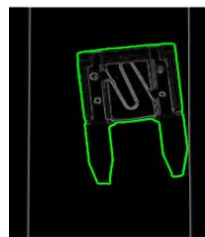


**Figure 6:** Filters (McAndrew, 2004)

- Discontinuity detection: locating where discontinuities are because of the object's borders (Figure 7)
- Edges: follows the edge of an object from a given point to locate the whole object (Figure 8)
- Connected components: follows the pixels that are next to each other on the object's border in order to detect the number of objects in the image (Figure 9)
- Cross correlation: making possible to compare two images to find its similarities (Figure 10)



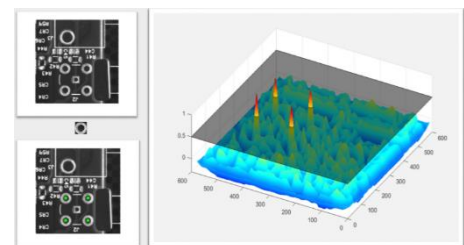
**Figure 7:** Discontinuity detection (Reina Terol, 2019)



**Figure 8:** Edge detection (Reina Terol, 2019)



**Figure 9:** Connected components (Reina Terol, 2019)



**Figure 10:** Cross correlation (Reina Terol, 2019)

The tasks that can be solved using computer vision can be categorized in geometry, positioning, motion, radiometry, spatial structure and texture, 3D modeling and higher-level tasks such as segmentation, object identification and classification or recognition and retrieval.

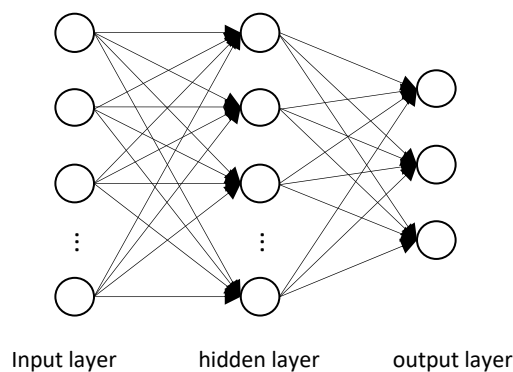
The applications of computer vision are as numerous as the tasks that can be executed. The most known application and also most developed nowadays is AI; with the aim of training computers for identification and classification of objects as independently as they could be (Jähne & Haussecker, 2000; SAS, 2019).

### 4.3 Artificial Intelligence: Deep Learning

Artificial Intelligence is an area of computer science that tries to get computers to imitate human-like intelligent behaviour, such as reasoning, adapting and self-correcting. For a system to be called artificially intelligent, according to the Turing test, it would have to be able to communicate in a natural language, to have knowledge and to store it somewhere, to do reasoning based on the stored knowledge, and to be able to learn from its environment (Kok, et al., 2009).

Looking at these requirements, it can be said that one of the most important branches of AI is machine learning. A system which is in an evolving environment must possess the ability to learn and to adapt to such changes to be called intelligent, this is done by using ANNs as it will be explained bellow. In other words, an intelligent system should be able to automatically extract an algorithm for the execution of a task based on existing accurate data, in order not to replicate this data, but to correctly predict new cases. That is the aim of machine learning (Ertel, 2017; Alpaydin, 2016).

The way AI and machine learning try to imitate human behaviour is by using ANNs. An ANN is based on the brain function and its inner communication. It is made up of artificial neurons connected among themselves and can reinforce or inhibit the activation of the neighboring neurons. The ANNs consist on three basic layers of artificial neurons as shown in Figure 11: An input layer exposed to the input signals that transmits information to the following layer, the hidden layer. In this layer the important features are extracted from the information received, and then transported to the output layer (Neapolita & Jiang, 2018; Deng & Yu, 2014).



**Figure 11:** Structure of an ANN

Depending on what kind of task is needed to be executed by the ANN, learning can be done in one way or another. The two main types of learning methods used in machine learning are the following: Supervised learning and unsupervised learning (Alpaydin, 2016; Neapolita & Jiang, 2018).

Supervised learning consists on training the system with an existing data set. The training set is made up of labeled data: inputs with their corresponding output. This kind of method is used with the goal of learning a mapping function from the input to the output, so that given new inputs, the system

can predict the correct output (Shukla, 2017). There are two basic categories in which supervised learning algorithms can be grouped:

- Classification: When the output variable is a category. For example, classifying emails or spam, identifying objects in an image, predicting gender based on handwriting.
- Regression: When the output variable is continuous. For example, predicting age, weight or salary.

There are several training methods in which supervised learning can be carried out. One of the most used due to its fast and easy training is transfer learning. Transfer learning consists on the modification of the last layers of an existing network with the aim of creating another one capable of working with different data than the original network.

Unsupervised learning, on the other hand, does not provide predefined labels for the input data, which means the system will not be given any training. The aim of this method is to find hidden regularities in the inputs, such as patterns, similarities or differences (Alpaydin, 2016; Bansal, 2017). It is mostly used in order to find existing categories in the input data given.

Deep learning is a sub-branch of machine learning that uses a specific kind of neural network architecture. The difference between deep learning networks and others is the number of hidden layers of the ANN used, since deep learning often uses more hidden layers than the older neural networks (Deng & Yu, 2014; Neapolita & Jiang, 2018). Some of the advantages it has over other approaches of machine learning are the large amount of data these neural networks can work on, solving problems directly to end instead of dividing them into sub-tasks, and even if it takes a longer time to train these networks, the testing time is reduced, increasing accuracy. Deep learning is being currently used in applications such as image and speech recognition, automatic text generation or automatic machine translation, which were difficult to implement with other types of techniques (Sultana, et al., 2020) .

## 5. Literature review

The field this project is based on has been researched many times before; in order to get an overview of the previously done work, this chapter analyses some of those documents for each part of the project.

### 5.1 Weed detection

Agriculture has always been an essential activity for survival. Over the last century, and more specific, over the last 15 years, agriculture has started to mechanise and digitise; due to this evolution and automation, labour flow was almost totally standardised. Nowadays, after introducing robotics and artificial intelligence into agriculture there is no need of standardization, robots are working collaboratively with humans and learning from them how to realize the basic agriculture tasks such as weed detection, watering or seeding (Marinoudi, et al., 2019).

Weed detection is one of those basic agriculture tasks that are being automatized and digitised, in this case, because of toxicity related to herbicides; so, reducing human intervention will make possible a decrease in the use of herbicides, increasing health care. To achieve this, robots able to detect plants and classify them into crop or weed are now introduced into agriculture (Dankhara, et al., 2019). This implementation has been done in multiples studies such as Dankhara, et al., (2019), where Internet of Things (IoT) is applied into an intelligent robot to differentiate crop and weed remotely; IoT is present in the communication between a Raspberry Pi, where the processing is done and the camera and sensors are connected, and the Data Server, where the Raspberry Pi sends the information obtained. This paper shows an accuracy of 90%-96% depending on if it is used a Convolutional Neural Network (CNN), a datasheet is being created or it is being used the training set.

Daman, et al., (2015) and Liang, et al., (2019) both introduce the use of automation into agriculture to identify weeds, and to do so, they make use of image processing techniques. Daman, et al., (2015) implement those techniques into an herbicide sprayer robot, capturing images from a Raspberry Pi camera and extracting pixels' colours to process them with diverse techniques in order to know whether it is a weed or not. Results were more than successful, after placing plants and weeds randomly, the robot was tested and weeds were almost totally identified and sprayed, taking the processing stage approximately 3 seconds. Liang, et al., (2019) implement image processing in drones instead of robots, that way, they not only detect weeds, but also monitor the growth of crops. By combining image processing and CNN in drones, they get different accuracies depending on the processing, which is from 98.8% with CNN to 85% using Histograms of Oriented Gradients (HOG).

All the previously mentioned processes can be done either in static by photos or in real-time by videos. Marzuki Mustafa, et al. (2007), have done a research about the implementation of a real-time

video processing. The crop is recorded and processed, offline, using various image processing techniques and a new developed algorithm that respond correctly to real time conditions. Finally, they achieved an accuracy over the 80%.

Not only the weed as a plant can be differentiated, more advanced studies such as Wafy, et al., (2013), differentiate the weeds seeds using Scale-Invariant Feature Transform (SIFT), an algorithm that extracts the interest points from an image; by using this technique, the minimum accuracy they have is 89.2%.

## 5.2 Image processing

There is no correct technique to process images in order to obtain the characteristics needed to identify their elements, and weed detection is not an exception. There are many papers where different techniques are shown. Olsen, et al., (2015) makes use of segmentation and a rotation variant of HOG in order to process the images and get the same illumination so they are robust to variations in rotation and scale. By using these techniques, they got an accuracy of 86.07%.

Samarajeewa (2013) compares two techniques: Local Binary Patterns (LBP) and  $L^*a^*b$  thresholding. LBP thresholds pixels intensity according to its surroundings; this way, only high intensity values are visualised, separating plants from the background.  $L^*a^*b$  thresholding selects a threshold value for each channel in RGB based on histograms. Then, in both techniques, erosion is applied to remove the noise that can have appeared. This procedure is done with RGB and HSV images; the results obtained show that LBP has an accuracy of only 48.75% whereas  $L^*a^*b$  thresholding has an accuracy of 89.96%.

Another technique usually used is Hough transform; Bah, et al., (2017) combines the Hough transform with simple linear iterative clustering (SLIC). This method focuses on the detection of crop lines; that way, what is not located in that line or differs from its neighbours, is supposed to be a weed. Firstly, the background is segmented and the shadows are eliminated; then the crop line is detected by using some operations that will end up in obtaining the 'skeleton' of the crop line, from that image, weed can be differentiated as said before. By following this method, it has been achieved an accuracy of more than 90% and an over-detection inferior to 2%.

Írías Tejeda & Castro Castro (2019) comes up with a generic Matlab algorithm for image processing of pictures with uniform illumination. The first step is a grayscale conversion with "rgb2gray" and green pixel subtraction from the converted image, in order to detect green plants in the images. Then filtering is done using "medfilt2", which applies a median filter for a neighbourhood of 3x3 pixels with the intention of noise reduction. Image thresholding follows using the Otsu method with the command "graythresh", in order to do thresholding segmentation to get the binarized image. Morphological reconstruction comes next, with "imfill" and "bwmorph" to fill the image regions and

holes. Next step is labelling and classification, where connected components are labelled with “bwlabel” and the smaller regions are removed since they are considered to be weeds. Finally, a threshold based on the classification values of the area for a crop or a weed is taken for further comparisons.

### 5.3 Deep Learning for weed and crop identification

Deep Learning neural networks range from deep neural networks, deep belief networks, recurrent neural networks and CNNs. The most usually used are CNN, whose layers apply convolutional filters to the inputs. The networks are rarely created from scratch and most of the ones used on projects are already existing networks such as LeNet, AlexNet, GoogleNet, SNET or CNET (Moazzam, et al., 2019).

Moazzam, et al., (2019) offers a summary of seven different studies, all of them use deep learning convolutional networks approaches for the weed/crop identification problem, as shown in *Table 1*. Even if all the papers mentioned focus on different types of crops, a common element is that most of them only focus on one crop. Studies using deep learning identification of multiple crops and weeds are not common.

|                                       | Deep Learning Type                            | Crop       | Training Setup   | Training Time                | Acquisition Setup     | Dataset Strength                  | Accuracy % |
|---------------------------------------|---|------------|--|------------------------------|-----------------------|-----------------------------------|------------|
| <i>Fawakherji, et al., 2019</i>       | Pixel wise segmentation using CNN             | Sunflower  | NVIDIA GTX 1070 GPU  | Three weeks                  | Nikon D5300 camera    | 500 images                        | 90         |
| <i>Knoll, et al., 2018</i>            | Image Based Convolutional Neural Networks     | Carrot     | GTX Titan having 6GB graphic memory                            | Not given                    | RGB CAMERA            | 500 images                        | 93         |
| <i>McCool, et al., 2017</i>           | Image Based Convolutional Neural Networks     | Carrot     | Not mentioned  | Not given                    | RGB CAMERA            | 20 training and 40 testing images | 90.5       |
| <i>Tang, et al., 2017</i>             | K-means feature learning accompanied with CNN | Soybean    | Not mentioned  | Not given                    | Canon EOS 70D camera  | 820 RGB images                    | 92.89      |
| <i>Miloto, et al., 2017</i>           | CNN based Semantic Segmentation               | Sugar beet | NVIDIA GTX1080Ti   | 200 epochs in about 48 hours | JAI AD-130 GE camera  | 10.000 plant images               | 94.74      |
| <i>Córdova-Cruzatty, et al., 2017</i> | Image Based Convolutional Neural Networks     | Maize      | Core i7 2.7 GHz 8 core CPU Computer with Nvidia GTX950M        | Not given                    | Pi camera Version 2.1 | 2835 maize and 880 weed images    | 92.08      |
| <i>Chavan, et al., 2018</i>           | AgroAVNET                                     | 12 classes | Intel Xeon E5-2695, 64GB RAM and NVIDIA TITAN Xp with 12GB RAM | Not given                    | RGB CAMERA            | 5544 images                       | 93.64      |

**Table 1:** CNN comparison (Moazzam, et al., 2019)



Starting with Fawakherji, et al., (2019), this study focuses on the classification of sunflower crops and weeds using pixel-wise segmentation with a CNN. With a training dataset of 500 images, the first step taken is the pixel-wise classification of soil and vegetation, using UNet semantic segmentation network. The second step is background removal and extraction of Regions of Interests (ROI) for their later classification in the third and final step as a crop or weed using a thirteen-layer CNN model. The accuracy obtained with this method is of a 90%.

Knoll, et al., (2018) and McCool, et al., (2017) both study the usage of image-based CNN for the detection of carrot crops and weeds. The first paper uses an eleven-layered network to classify three categories: weed, carrots and background. The network is trained with 500 RGB images taken with a camera. As for the second paper, it uses GoogleNet pretrained on ImageNet and compresses it creating a deep CNN which is then trained on an online dataset. This method reported an accuracy of 90.5%, meanwhile the first paper reported an accuracy of 93%.

For soybean classification Tang, et al., (2017) uses k-mean classification pre-training prior to the CNN training. The CNN used consists of a ten layered convolutional network trained with a dataset of 820 RGB images to classify between soybean and three different types of weeds. The accuracy of this process is of a 92.89%. A similar accuracy percentage is found in the classification of maize crops and weeds, for this, Córdova-Cruzatty, et al., (2017) uses approximately 3600 maize and weed images taken by a Raspberry Pi 3 camera, and performed the testing on four CNNs: LeNet, AlexNet, SNET and CNET. The best accuracy obtained was with CNET, with a value of 92.08%.

Miloto, et al., (2017) focuses on sugar beet and weed classification. With a 94.74% of accuracy, the training performed on the semantic segmentation-based CNN was done for 48 hours, using nearly 10,000 images. The last paper, Chavan, et al., (2018) is the only one that tries the classification of multiple crops, creating a hybrid version of AlexNet and VGGNET for weed and crop classification: AgroAVNET, which is a CNN of five layers, trained with 5544 images, with an accuracy of 93.64%.

In conclusion, crop and weed detection with the use of deep learning is not yet a usual topic of research, even if there are more and more attempts. There are still many research gaps not considered like the differentiation of different crops and weed combinations. Furthermore, even some major essential crops are lacking in this kind of investigation, as there is still a need of creating big datasets for these crops. Deep learning is still a new a tool for the autonomous agricultural applications, yet it seems to be a promising technique and more accurate than other approaches (Moazzam, et al., 2019).

From these researches the needed knowledge about the necessary pre-processing techniques that will be used in this project has been acquired; some of these are filtering, binarization and histograms, a deeper study on them will be done during the development to make sure they suit correctly. Also, through the study of ANNs, some projects using CNNs have been found, being one of those nets AlexNet, the one chosen for this project; by this research, a vision on how to work with these nets has been acquired, as well as the accuracy expected in this kind of projects.

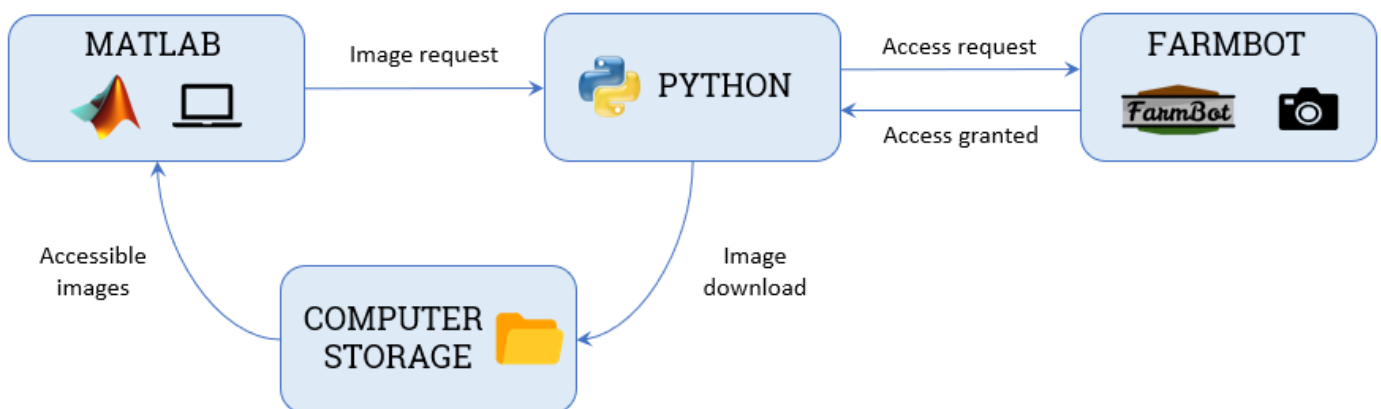


## 6. Initial design

In this chapter, both an overview of the initial design of the project and the first prototype developed are explained. The relevant modifications for the design of the final project and the final project itself are explained in more detail in the following chapters.

### 6.1 System overview

As it has been previously stated, the main focus of this project is the training of a neural network in order to differentiate crops and weeds. To achieve this, Matlab is used as the principal programming software for the image processing and the network training. Other software such as Python and FarmBot are needed to help with the acquisition of images. To store the pictures, the computer storage is used as well. The following image (*Figure 12*) shows a representation of how these software are connected:



**Figure 12:** System overview

In *Figure 12*, each block has its own functionality, being all of them crucial to the development of the project. To have a deeper understanding of each point, they are explained below.

- *Matlab* is the principal software of the system. It takes images as its input, processes them to improve their characteristics and finally trains and evaluates an ANN with those images; the ANN correctly trained is the output of the program. The trained network will be later used to detect crop and weed from any picture.

The images used are the ones captured by FarmBot; to make this possible, the first step of the Matlab programming is the acquisition of those images by executing a Python code.

- *Python* works as the intermediary between Matlab and FarmBot. Once it is executed, it connects to FarmBot REST API and requests access to download the pictures taken by the robot. After the access is granted, the Python code downloads those pictures into the computer where Matlab is running in order to make them accessible.
- *FarmBot* is the only source of image capturing for this project. The robot is programmed on its UI in order to take the needed pictures, which are stored waiting for the python code request to access them. FarmBot is also used as a comparison source for the thesis results, comparing the accuracy and characteristics of its own weed detector software with the one developed.
- All the downloaded images are stored in the *Computer Storage*, where they will be accessible for Matlab to retrieve them.

## 6.2 Prototype

The goal of the initial design was obtaining a trained network to later modify, if needed, in order to obtain accurate results. These accurate results would be a correct distinction between crops and weeds in the images captured by the FarmBot.

The project has been developed using three different software, therefore there are three different codes that, linked together, conform the final project. For the initial design of the project, a prototype for each code has been created accordingly to the expected function of each software.

The programs chosen were the FarmBot software, Matlab and Sublime Text 3 as a Python programming environment. The FarmBot UI was used because it is the programming interface from where the robot and the camera used can be controlled. As the pictures taken are stored in the FarmBot REST API, to access them easily Python was chosen following the advice from this thesis supervisor and some posterior research. Finally, Matlab was chosen as the main program because of previous experience working with this software, and the great amount of available information resources. The full code developed in each of these programs can be found in *Appendix A*.

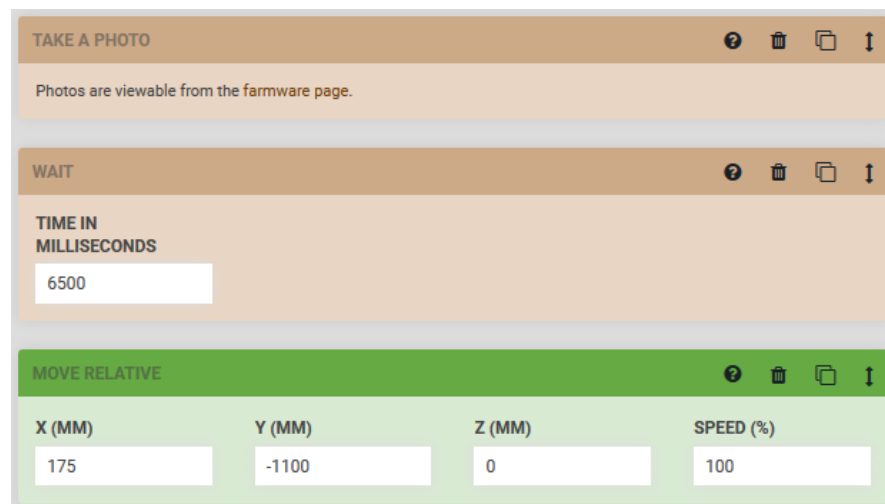
To choose the pre-created ANN for this project, a study was carried out in the literature review. AlexNet was the chosen network, due to the accuracy shown in various projects similar to this thesis. From the literature review it was also decided that image classification was a suitable method for the identification of crops and weeds.

Finally, for the network training, the pictures used were not only from the FarmBot but also from pictures taken onsite with other cameras. Those pictures were used to prevent a lack of time while developing the project, in fear the crop or weeds would not grow on time.

### 6.2.1 Image acquisition

This part is carried out in two programs, the FarmBot UI and Sublime Text 3. The first one is used to take the pictures and the second one to download them into the computer. In this point, an overview of the most important parts of both codes is explained; the full code explained in detail can be found in *Appendix A*.

The first step is to take pictures of the whole field. This is done with twenty-five pictures, five for each column of the five rows that the FarmBot workspace consists on. To do so, FarmBot UI has commands called 'Take Photo' and 'Move To', by adding these commands (*Figure 13*) into a sequence the robot moves to the specified points and then takes pictures of the crops. Once each picture is taken, it is automatically sent to its Google Cloud Server, from where all the pictures will be downloaded.



**Figure 13:** Sample of FarmBot code

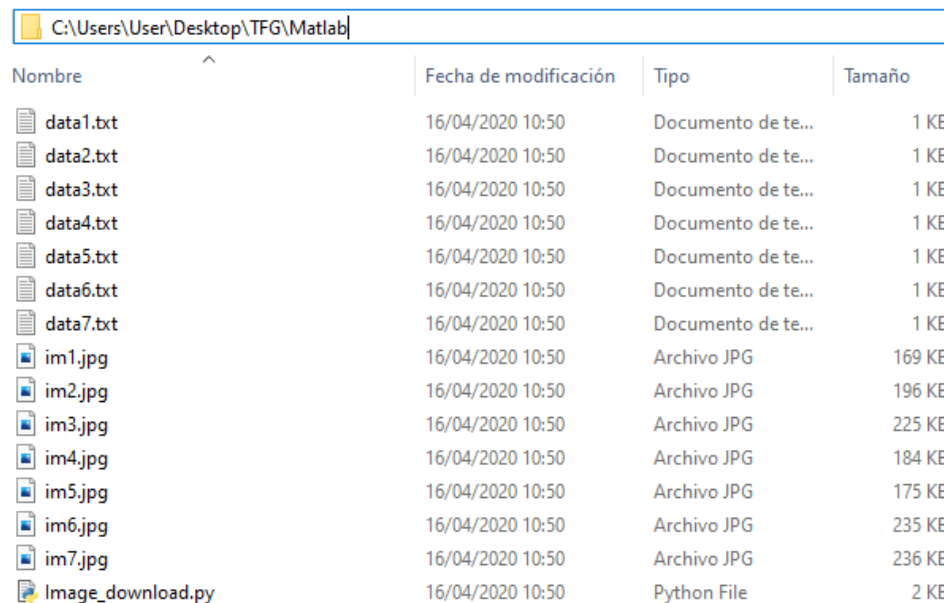
After the pictures are stored by FarmBot, it is necessary to access them. By using Sublime Text 3 as programming environment, a Python code has been developed with the aim of downloading and storing them into the computer so Matlab can later make use of them, as shown in *Figure 14*.

```
# Number of images
numel = len(images)

# Download data and images
for i in range(0, numel-1):
    image_url = images[i]['attachment_url']
    image_data = images[i]['meta']
    myfile = requests.get(image_url)
    open(path + 'im' + str(i+1) + '.jpg', 'wb').write(myfile.content)
    open(path + 'data' + str(i+1) + '.txt', 'wt').write(json.dumps(image_data))
```

**Figure 14:** Downloading images code

Previously to *Figure 14* code, access has been granted and the images have been requested. In order to download them, the number of available images is counted and, one by one, their URL and their coordinates are stored. Finally, a request to download what is addressed on each URL is made, and the image and coordinates are stored in the computer as files with unique names, as shown in *Figure 15*.



| Nombre            | Fecha de modificación | Tipo               | Tamaño |
|-------------------|-----------------------|--------------------|--------|
| data1.txt         | 16/04/2020 10:50      | Documento de te... | 1 KB   |
| data2.txt         | 16/04/2020 10:50      | Documento de te... | 1 KB   |
| data3.txt         | 16/04/2020 10:50      | Documento de te... | 1 KB   |
| data4.txt         | 16/04/2020 10:50      | Documento de te... | 1 KB   |
| data5.txt         | 16/04/2020 10:50      | Documento de te... | 1 KB   |
| data6.txt         | 16/04/2020 10:50      | Documento de te... | 1 KB   |
| data7.txt         | 16/04/2020 10:50      | Documento de te... | 1 KB   |
| im1.jpg           | 16/04/2020 10:50      | Archivo JPG        | 169 KB |
| im2.jpg           | 16/04/2020 10:50      | Archivo JPG        | 196 KB |
| im3.jpg           | 16/04/2020 10:50      | Archivo JPG        | 225 KB |
| im4.jpg           | 16/04/2020 10:50      | Archivo JPG        | 184 KB |
| im5.jpg           | 16/04/2020 10:50      | Archivo JPG        | 175 KB |
| im6.jpg           | 16/04/2020 10:50      | Archivo JPG        | 235 KB |
| im7.jpg           | 16/04/2020 10:50      | Archivo JPG        | 236 KB |
| Image_download.py | 16/04/2020 10:50      | Python File        | 2 KB   |

**Figure 15:** Downloaded and stored images

As it can be seen in *Figure 15*, all the pictures FarmBot has taken are successfully downloaded and stored. The main problem that can be found is that by putting no limit to the number of pictures this program is downloading, there is no control over the images that are being stored, including also older pictures that are not valid anymore. This will be fixed in the final project.

## 6.2.2 Image classification

Image classification is done in Matlab. The aim of the Matlab code is to train a pre-created network in order to make it able to perform a classification between spinach and weeds. The full code explained in detail can be found in *Appendix A*, while the training and classification processes, the most important points, are discussed below.

Before starting to work with Matlab, there is one more step to perform. To work with the downloaded images, Matlab needs to know exactly what is inside of every picture. This has to be done manually by separating each downloaded picture in folders, named accordingly to the different categories the net is going to differentiate. Once this is done, Matlab accesses these folders and randomly divides the images in three groups: training, validation and testing, in order to work with them separately.

The training process is shown in *Figure 16*. It consists in the modification of AlexNet by changing its last two layers, and then trains it on the training set of images with some options that will determine how the network will learn. These steps are followed due to the selection of Transfer Learning as the network's learning method.

```
%% Net modification:

net = alexnet;
layers = net.Layers;
layers(23) = fullyConnectedLayer(2);
layers(25) = classificationLayer;
opts = trainingOptions('sgdm','InitialLearnRate',0.001, 'ValidationData',valids, 'Plots', 'training-progress');
[trainedNet, info] = trainNetwork(trainds, layers, opts);
```

**Figure 16:** Network training

With the training options it is possible to visualize the training progress. In the image below two graphs can be observed, the first one accuracy – iteration and the second one loss – iteration. Here a new concept is included, the ‘epochs’. An epoch is a training cycle with all the images, those images are divided into smaller batches that will be introduced to the network while training, the number of batches in an epoch is the number of iterations.

Figure 17 shows the training progress of our network. The training for this prototype consists on 30 epochs with one iteration each. For each iteration, a new value is added to the graphs, the accuracy and the loss. Accuracy is the percentage of correctly classified elements while loss is how badly the network performs on the training data. Validation can also be seen in this picture, as a black dotted line, at the beginning and at the end, in order to control that the network keeps improving and stops if it starts getting worse.

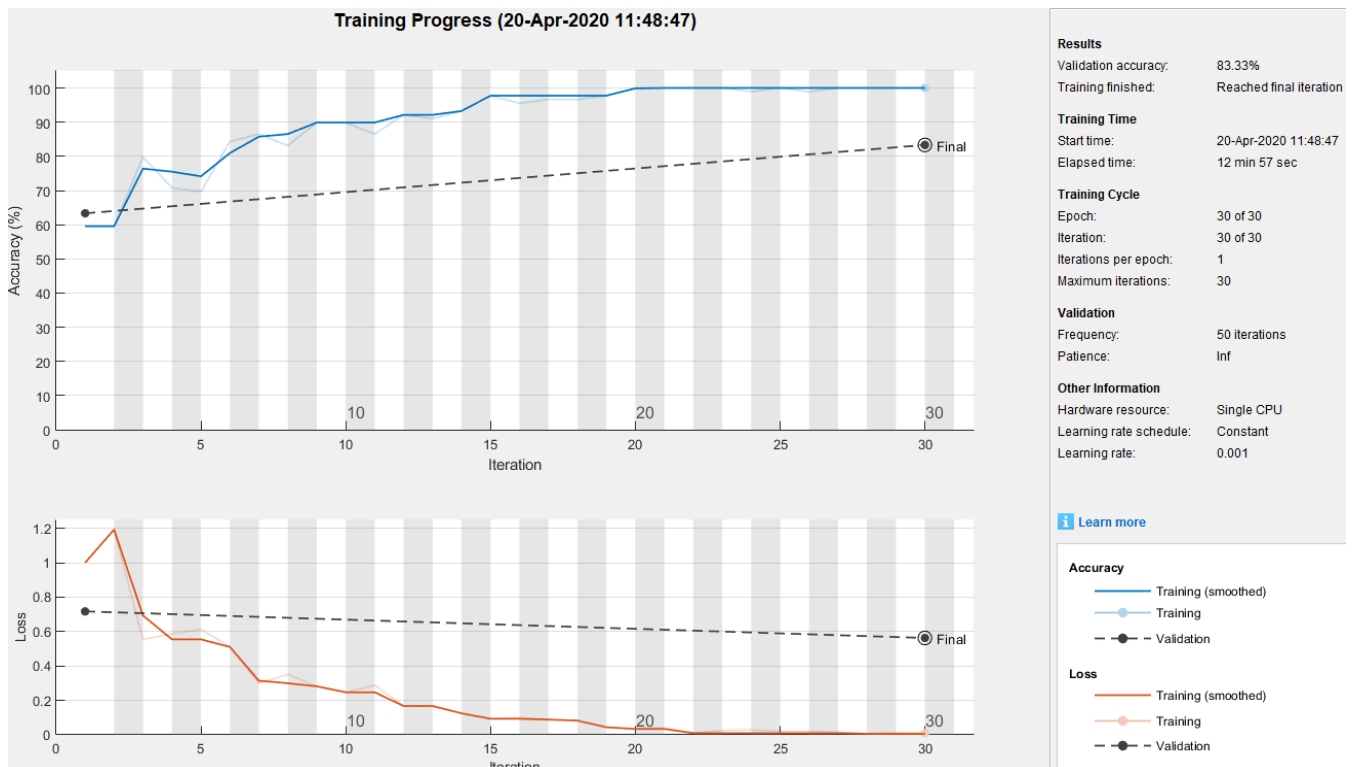


Figure 17: Training progress

When the network is finally trained, the percentage of accuracy and loss can be seen as the training results as it is shown in Figure 18. The training done in this prototype has taken about 13 minutes and has an validation accuracy of 83.33%. This result is good enough to test the network with new images and get the final accuracy.

```
>> [trainedNet, info] = trainNetwork(trainds, layers, opts)
Training on single CPU.
Initializing input data normalization.
```

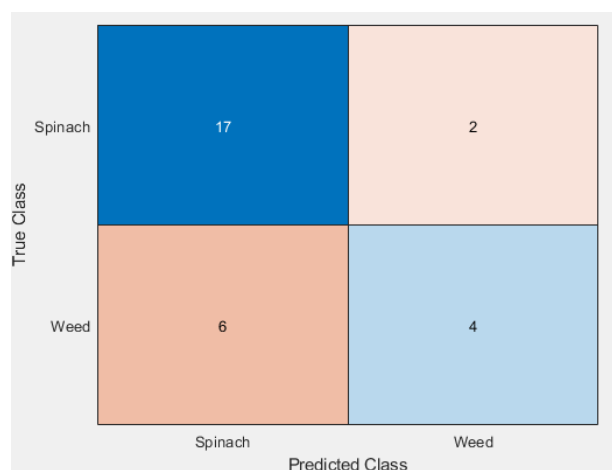
| Epoch | Iteration | Time Elapsed<br>(hh:mm:ss) | Mini-batch<br>Accuracy | Validation<br>Accuracy | Mini-batch<br>Loss | Validation<br>Loss | Base Learning<br>Rate |
|-------|-----------|----------------------------|------------------------|------------------------|--------------------|--------------------|-----------------------|
| 1     | 1         | 00:00:34                   | 59.55%                 | 63.33%                 | 1.0006             | 0.7177             | 0.0010                |
| 30    | 30        | 00:12:57                   | 100.00%                | 83.33%                 | 0.0099             | 0.5632             | 0.0010                |

Figure 18: Training results

The evaluation of the network is the final part of this prototype. By introducing the test images as it is done in *Figure 19*, the network is tested and the real accuracy of this classification is obtained. The final accuracy is of 72.41%. In order to know where the network fails, a confusion chart is done as it can be seen in *Figure 20*. From this chart, the number of images correctly and wrongly classified can be observed. As it is seen, the main problem of this network is the classification of weeds because it mistakes 6 out of 10 with spinach crops, while the spinach crop is mostly well classified.

```
>> Preds = classify(trainedNet,testds);  
>> accu = nnz(Preds == testImgs.Labels)/numel(Preds)  
  
accu =  
  
0.7241
```

**Figure 19:** Classification and evaluation of the network







**Figure 20:** Confusion chart

The performance of this prototype is good in terms of detection accuracy as it has a final accuracy of 72.41%, but it can be improved. First of all, to simplify the prototype, two different types of weed have been labeled as the same one; for the final project both types of weed need to be classified separately. Another aspect to consider is that, in order to get more precise results, a pre-processing can be done to the pictures so some of the most important features can be emphasized, also adding more images to the whole process will reflect in better results. Last but not least, image classification implies having only one kind of plant in each picture, so it is needed to manually crop and separate the weeds and the spinachs in each picture taken. This can be solved by using 'object detection' which is done with Regions with Convolutional Neural Networks (RCNN) instead of a CNN. All the final changes will be taken into account in the final project, *Chapter 8*.

## 7. Image processing

The main objective of this thesis is training a neural network to perform the differentiation between certain types of crop and weeds. To do so, the network is trained with images taken using a FarmBot. In order to have the best accuracy possible, the images are pre-processed so the most differentiable features are highlighted and the best quality possible is achieved. In this chapter, a research on some pre-processing methods is done, and the most suitable ones are chosen for the project.

In terms of image quality, not only their processing is important, but also the camera specifications play a big role. Depending on the camera, images have different quality. For this project, four different cameras have been tested in order to use the one where the crops and weeds are more visible. Those cameras were: FarmBot cameras versions 1.3 (*Figure 21*) and 1.5 (*Figure 22*) and 'C925e webcam' from Logitech (*Figure 23*) and 'Xiaomi Redmi Note 8 Pro' camera from Xiaomi (*Figure 24*). A comparison of those cameras specifications and photos is made bellow in *Table 2*.

|                           | FarmBot Genesis camera<br>v1.3  | FarmBot Genesis camera<br>v1.5  | C925e webcam Logitech  | Xiaomi Redmi Note 8 Pro<br>camera  |
|---------------------------|---|---|--|--|
| <i>Resolution</i>         | 1024x768  | 1024x768  | 1920 x 1080  | 4624x3472  |
| <i>Focal<br/>Distance</i> | 1 m   | 0.5 m   | Not defined  | Not defined  |
| <i>Picture</i>            |  <p><b>Figure 21:</b> Picture taken with FarmBot Genesis camera v1.3</p> |  <p><b>Figure 22:</b> Picture taken with FarmBot Genesis camera v1.5</p> |  <p><b>Figure 23:</b> Picture taken with C925e Logitech webcam</p> |  <p><b>Figure 24:</b> Picture taken with Xiaomi Redmi Note 8 Pro camera</p> |

**Table 2:** Cameras comparison

The aim of the camera comparison was to choose the one that takes the photos where crops and weeds are more noticeable. By looking at the table above, a picture taken with each camera can be observed. *Figure 21* is a picture taken with FarmBot Genesis camera v1.3, where it is difficult to differentiate anything because of the brightness. *Figure 22* is taken with a different FarmBot version, v1.5, here the plant is easy to recognize but still it has a lot of brightness making difficult the differentiation of different plant species. In *Figure 23*, taken with an external USB camera attached to the robot, crops are differentiable and less bright, still needing image processing to achieve better quality. Last but not least, in *Figure 24* a picture taken with a Xiaomi Redmi Note 8 Pro mobile phone



is shown; the pictures taken with this camera will only be used to train the network even though their quality is the best because the phone cannot be attached to the robot.

This thesis is finally done using the pictures taken with Logitech's camera attached to the robot and Xiaomi Redmi Note 8 Pro camera in order to get a bigger dataset. To get the best accuracy possible differentiating crops and weeds from the pictures taken, a pre-processing is done to them. Four different processing methods, some of them found on this project literature review, are tested with *Figure 23* to find the most optimum pre-processing techniques to implement. Those methods are background removal, histogram equalization, sharpening and glare removal.

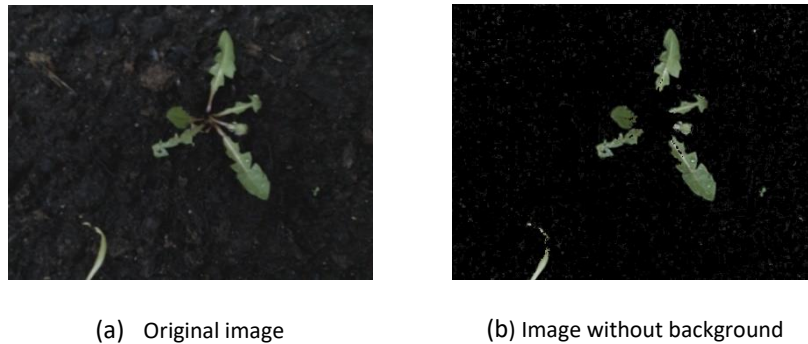
## 7.1 Background removal

Background removal is done to extract objects from a picture by eliminating the background. It puts some restrictions on the pixels based on their saturation and hue characteristics. If the pixels meet those characteristics then they are not part of the background and will appear on the picture, if they do not meet them, they will be eliminated.

```
3 - im = imread('C:\Users\User\Desktop\TFG\Matlab\FarmBot\im10.jpg');  
4  
5 - imhsv = rgb2hsv(im);  
6 - bH = imhsv(:,:,1)>0.2 & imhsv(:,:,1)<0.5;  
7 - bS = imhsv(:,:,2)>0;  
8 - imb = bH.*bS;  
9 - imGreen(:,:,1) = double(im(:,:,1)).*imb;  
10 - imGreen(:,:,2) = double(im(:,:,2)).*imb;  
11 - imGreen(:,:,3) = double(im(:,:,3)).*imb;  
12 - imGr = imGreen/255;
```

**Figure 25:** Background removal code

The Matlab code for background removal is shown in *Figure 25*. It takes an image 'im' and binarizes it, then it thresholds the saturation and hue of every pixel to extract only the ones close to green colour. The final image is conformed only with the pixels that meet those conditions. An example of background removal is found in *Figures 26*, it is easy to observe the difference between the original picture (*Figure 26 (a)*) and the processed one (*Figure 26 (b)*). As it is seen, the performance of this technique does not work as well as expected in this project because it detects part of the weeds as background and, sometimes, it does not detect the weed at all.



**Figure 26:** Background removal

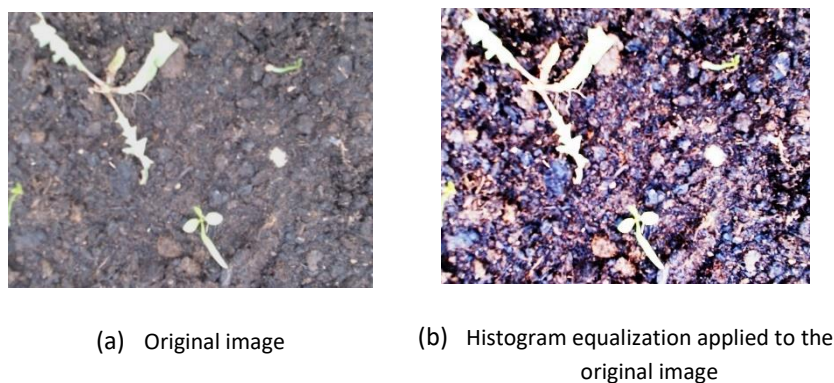
## 7.2 Histogram equalization

As said in *Section 4.2*, a histogram shows how many times a gray level appears in an image. By using histogram equalization an image is modified to get the optimum usage of its color scale. When a picture is mostly dark or bright, it is because its histogram is situated either too close to the left or to the right. Histogram equalization ‘extends’ the picture’s histogram in order to make it possible for the image to get a higher number of different colour levels.

In Matlab, histogram equalization is done with the command ‘histeq’ as shown in *Figure 27*. The image obtained as a result is displayed on *Figure 28 (b)* compared to the original one in *Figure 28 (a)*. As it can be seen, with the pictures used in this thesis the result is not good. This processing worsens the quality of any picture with good brightness, making it difficult to differentiate among crops and weeds.

```
imeq = histeq(im);
```

**Figure 27:** Matlab's command for histogram equalization



**Figure 28:** Histogram equalization

### 7.3 Sharpening

Sharpening is done to improve the content of a blurry picture. It increases the contrast between different colors of an image, making the edges where these different colors meet stand out. This way, the shapes of the figures present in the image are more defined. In Matlab, this is done with 'imsharpen', as shown in *Figure 29*.

```
im = imsharpen(rgbimage, 'Radius', 2, 'Amount',1); % Sharpening
```

**Figure 29:** Image sharpening in Matlab

This command, 'imsharpen', takes a colored image and sharpens it, taking into account parameters such as 'Radius' and 'Amount'. The 'Radius' is in control of how much of the region around the edges should be sharpened, and the 'Amount' controls the contrast increase level, making it softer or stronger depending on the value introduced. *Figure 30* shows an example image taken from the pictures captured by FarmBot (*Figure 30 (a)*), and the results of applying the sharpening to it with different values of each parameter *Figures 30 (b)* and *(c)*.



(a) Original image



(b) Sharpened image with radius 1  
and amount 2



(c) Sharpened image with radius 2  
and amount 2

**Figure 30:** Sharpening

### 7.4 Glare removal

The final processing alternative is glare removal which, as the name indicates, aims to reduce the white bright parts of the pictures caused by overexposure. This is done with a set of commands as shown below in *Figure 31*.

```
im = readimage(ds,i);  
imgray = rgb2gray(im);  
bin = imgray > 225;  
binary = imdilate(bin, true(1));  
red = im(:,:,1);  
green = im(:,:,2);  
blue = im(:,:,3);  
  
red = regionfill(red, binary);  
green = regionfill(green, binary);  
blue = regionfill(blue, binary);  
  
rgbimage = cat(3, red, green, blue);
```

**Figure 31:** Glare removal code

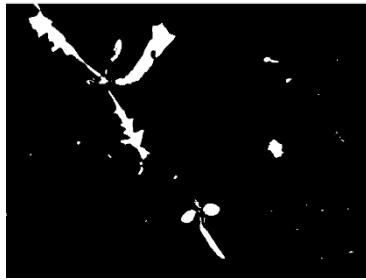
The code first loads the image in a variable called 'im' to then convert the original photo into a grayscale image (*Figure 32 (b)*). The next step is creating a binary picture taking only those pixels that have a value above 225 from the grayscale image, which are the brighter and whiter pixels of the original image (*Figure 32 (c)*). This picture is then dilated with 'imdilate' to fill the small spots near the whiter area. The three channels of the original RGB image are separated, and each of them is filled with the 'regionfill' command, using the binary image as a mask. This command fills the regions specified by the white pixels of the binary image by replacing those pixel values with the neighbouring pixel values. Finally, the resulting matrices are concatenated with the 'cat' command in order to recreate the RGB image. The resulting picture is shown in *Figure 32 (d)*, where it is possible to observe how the white pixels of the original image (*Figure 32 (a)*) have now become less bright and are colored according to their surroundings.



(a) Original image



(b) Grayscale image



(c) *Binary image*



(d) *Resulting image*

**Figure 32:** *Glare removal*

After studying these four possible ways of processing the images, it was decided that only sharpening and glare removal would be used. This was done because histogram equalization does not work well with images with good lighting, and background removal transforms the image too much from the original picture. Glare removal and sharpening, on the other hand, are essential for this project since, due to camera settings and the surrounding light, the images appear blurry and with overexposure. *Figure 33 (b)* shows an example of an image that has gone through this processing comparing to the original one in *Figure 33 (a)*.



(a) *Original Image*



(b) *Processed Image*

**Figure 33:** *Final processing*



## 8. Project development

The development of the final project has been done based on the prototype previously explained in *Chapter 6*. As the prototype results were not as good as they could have been, some changes have been done to it in order to achieve better results, these modifications lead to the final project and it is going to be explained in this chapter, whereas the final project code can be found on *Appendix B*.

The modifications, in order to have an overview of all the changes, are the following:

- The number of photos taken by the FarmBot has increased from 25 to 54.
- In Python, a command to erase older photos has been added in order to work only with the latest pictures.
- In Matlab there have been two big changes: the introduction of pre-processing techniques to the pictures in order to improve their characteristics, and a change of network from a CNN to a RCNN to perform object detection instead of image recognition; therefore, all the Matlab code has been updated.

The aim of this thesis is to implement a weed detection system able to differentiate between crops and weeds using neural networks. To make the project more specific, a crop and two different weeds have been chosen, spinach as crop and dandelions and cleavers as weeds; these can be seen in *Figure 34*. This means that the neural network of this project will differentiate spinach (circle 2), dandelions (circle 1) and cleavers (circle 3).



**Figure 34:** Crop and weeds chosen for this project

The structure of the project is the same as in the prototype. It has three phases: the picture taking done with FarmBot's software, the image download done in Python and, finally, the image processing and network training done in Matlab. Every part is going to be explained taking into account the modifications previously mentioned.

## 8.1 FarmBot

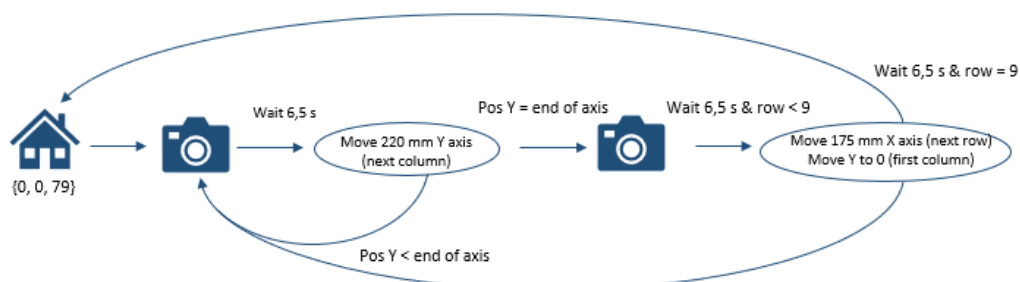
FarmBot is the main source of picture taking, by using the Logitech camera attached to the robot, it is programmed on its UI in order to take photos of the field. Every picture has dimensions of 385x280 mm and the field 1575x1320 mm (*Figure 35*), therefore 54 photos are taken.



**Figure 35:** FarmBot's field dimensions

The number of pictures taken has been increased from the prototype, then the number of movements the robot must do has also changed. In the programming this is reflected as an increment on the number of different positions the robot has to achieve, and as consequence, the number of commands sent to it.

The code follows almost the same sequence as before, changing just the number of times it repeats each sequence and the distances it moves (*Figure 36*). The robot is sent home to {0, 0, 79} where 79 mm is the height from where all the pictures are going to be taken. Then, until the robot arrives to the end of the first row (Y axis), it executes five times the picture taking and moves to the next position in that axis. When it arrives at the end of the axis, a last picture is taken, and it moves to the beginning of the next row. This is done nine times, moving over the whole field and taking pictures of it. When it arrives to its last position, it takes the last photo and moves straight to the home position again, ready to start over.



**Figure 36:** FarmBot's sequence diagram

Once each picture is taken, it is uploaded to FarmBot's Google Cloud storage. In order to work with those images, it is necessary to access that storage. In this thesis this issue is addressed by requesting its access through a Python code.

## 8.2 Python

A Python code developed in Sublime Text 3 interface is the link between the two main software of this thesis, FarmBot and Matlab. Its function is to ask the FarmBot storage for access to the pictures and then download them in the computer so Matlab can use them.

The Python code written for the final thesis is much similar to the one in *Section 6.2.1*. For the training part of the network, the images needed are acquired the exact same way as in the prototype. Once the network is correctly trained and functional, to detect the plants present in the images only one set of images is needed, representing the FarmBot bed where the crops are planted. Because of this, a slight modification is done to the code, downloading only the 54 most recent images as shown in *Figure 37*.

```
33 # Download data and images
34 for i in range(0, 53): # From the latest picture (0) to picture 54
35     image_url = images[i]['attachment_url']
36     image_data = images[i]['meta']
37     print(image_url)
38     print(image_data)
39     myfile = requests.get(image_url) # Download the picture
40     open(path + 'im' + str(i+1) + '.jpg', 'wb').write(myfile.content) # Save the picture
41     open(path + 'data' + str(i+1) + '.txt', 'wt').write(json.dumps(image_data)) # Save the picture data
```

**Figure 37:** Image download in Python

Keeping in mind that the plant detection is not supposed to be done with only one set of images, it was needed to add to the python code a few lines (*Figure 38*) in order to delete the images stored in the computer to clear the folder and download more recent pictures to detect. This is done after accessing the FarmBot images and before downloading them. Once the images are correctly stored in the specified folder, Matlab is where they are processed and used for the training and detection.

```
28 files = 'C:/Users/User/Desktop/TFG/Matlab/FarmBot'
29 filesToRemove = [os.path.join(files,f) for f in os.listdir(files)]
30 for f in filesToRemove:
31     os.remove(f)
```

**Figure 38:** Deleting Images with Python



## 8.3 Matlab

As Matlab is the core programming environment for this project, it is the most different from the initial design shown in *Chapter 6*. The main change to highlight is the switch of approach for the plant recognition from image classification to object detection. The prototype was based on taking a picture and classifying it, object detection on the other hand studies the contents of the image and recognizes and classifies each element. To do this the concept of RCNN is introduced.

RCNNs work different than CNNs in a way that it first extracts region proposals from the given images, and then classifies each region as a CNN would do with a normal image. The final result of this detection is a bounding box corresponding to the object location within the image, as well as its associated label, which determines the class the detected object should belong to.

In this subchapter, a quick update on the Matlab code is presented to understand how the program has been developed taking into consideration all the changes made. This update involves the image processing, the network training and the plant detection.

### 8.3.1 Image Processing

The image processing is fundamental to improve the quality and conditions of the input images. In *Chapter 7*, different processing techniques were considered and thoroughly explained, in order to decide which ones should be applied to the pictures obtained. After studying the four possibilities, it was decided to choose only two of the proposed techniques: glare removal and sharpening. These two were chosen to remove bright white light present due to overexposure, and to emphasize the blurry edges of shapes and objects of the images. The picture below (*Figure 39*) is an excerpt of the final Matlab code applicable to both the training and detection set of images.

```
11 - for i = 1:1:num
12 -
13 -     im = readimage(ds,i);
14 -     imgray = rgb2gray(im);
15 -     bin = imgray > 225;
16 -     binary = imdilate(bin, true(1));
17 -     red = im(:, :, 1);
18 -     green = im(:, :, 2);
19 -     blue = im(:, :, 3);
20 -
21 -     red = regionfill(red, binary);
22 -     green = regionfill(green, binary);
23 -     blue = regionfill(blue, binary);
24 -
25 -     rgbimage = cat(3, red, green, blue);
26 -     im = imsharpen(rgbimage, 'Radius', 2, 'Amount', 1.65);
27 -
28 -     filename = sprintf('%s_d.%s', 'C:\Users\linac\Desktop\TFG\Processed_Images\processed_', i, 'jpg');
29 -     imwrite(im, filename);
30 -
31 - end
```

**Figure 39:** Image processing

### 8.3.2 Network training

To train the network, some images are introduced to it with their proper identifiers so the net can learn from them. 227 photos have been taken to train the network and test its performance. To do this, in Matlab, the datastore where the processed images are, is split in two different groups as shown in *Figure 40*: 'imTrain' and 'imTest'. In this thesis, the set of images is randomly divided into imTrain with 159 images and imTest with 68.

```
41 %% Splitting Datastores
42
43 - imds = imageDatastore('\\\\studentnas1\\studenthome\\2019\\bl91auma\\Desktop\\TFG\\Processed_Images','Labelsource', 'foldernames');
44 - [imTrain, imTest] = splitEachLabel(imds, 0.7, 'randomized');
45
```

**Figure 40:** Splitting the images datastore

As said in this chapter's introduction, to implement object detection the network used in this thesis is a RCNN. In order to make the network able to detect certain objects in pictures, it is necessary to train it with pictures where those objects are defined and surrounded. This is done manually using ImageLabeler application in Matlab, obtaining a table with the location of all the objects in each picture called 'ground truth'. This table is done for each set of images: training, test and prediction.

One of the delimitations of this thesis was that the network used is not created from scratch, this means that a pre-created network is going to be modified. This method of working with neural networks is known as transfer learning. In Matlab it is implemented as shown in *Figure 41* by loading a network and modifying two out of three of its final layers.

```
46 %% Network modification
47
48 - net = alexnet;
49 - layers = net.Layers;
50 - layers(23) = fullyConnectedLayer(4);
51 - layers(25) = classificationLayer();
52
```

**Figure 41:** Network modification

As there are multiple ways of training the network, some options are defined in advance as in *Figure 42*. In this project the options chosen are the method, mini batch size, maximum number of epochs and initial learning rate. The method chosen is 'Adam' (Adaptive Moment Estimation) due to its good performance in several trials. The mini batch size is how many images are introduced to the network while training, in this case 64. The maximum number of epochs is 20, meaning the whole training data is used 20 times. To set how quickly the network learns, the initial learn rate is defined as  $3 \cdot 10^{-4}$ , an average learning speed used in similar projects. The selection of these options is explained in detail in *Appendix B*.

```
53 %% Training options
54
55 - options = trainingOptions('adam', 'MiniBatchSize', 64, 'MaxEpochs', 20, 'InitialLearnRate', 3e-4);
56
```

**Figure 42: Training options**

Finally, the net is trained with the options and the ground truth table. *Figure 43* shows the Matlab command to train the object detector. The network training has three different parts: the extraction of region proposals, the training and the bounding box regression training. The training is printed as shown in *Figure 44*, with each of its phases differentiated. *Figure 44 (a)* is the extraction of region proposals, in this step, the objects previously labelled in each image are detected and extracted so the network can process and learn from them. The training itself is *Figure 44 (b)*, where its parameters are shown. The training images are introduced to the net as mini batches, with their own train accuracy, loss and the amount of time elapsed for their processing; this is done iterating the same process until all the training images are used and another epoch starts. Finally, a bounding box regression is done as in *Figure 44 (c)*, detecting the objects and its boxes and labels as in the beginning in order to refine its detection and the network accuracy.

```
57 %% RCNN Training
58
59 - [rcnn,info] = trainRCNNObjectDetector(traindata, layers, options);
60
```

**Figure 43: RCNN training command**

```
*****
Training an R-CNN Object Detector for the following object classes:

* Cleaver
* Dandelion
* Spinach

--> Extracting region proposals from 159 training images...done.
```

(a)

```
--> Training a neural network to classify objects in training data...

Training on single CPU.
Initializing input data normalization.
=====
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Base Learning |
|       |          | (hh:mm:ss)  | Accuracy   | Loss       | Rate         |
|=====|=====|=====|=====|=====|=====|
| 1     | 1       | 00:00:03    | 12.50%     | 3.7922     | 0.0003       |
| 1     | 50      | 00:02:59    | 79.69%     | 0.6162     | 0.0003       |
```

(b)

```
|      20 |      4050 |      04:00:50 |      98.44% |      0.1167 |      0.0003 |
|      20 |      4100 |      04:03:46 |      93.75% |      0.1468 |      0.0003 |
|=====|

Network training complete.

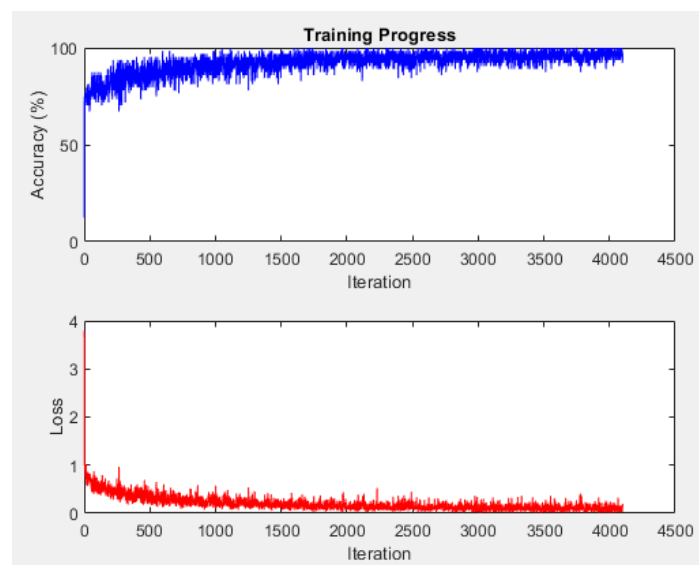
--> Training bounding box regression models for each object class...100.00%...done.

Detector training complete.
*****
```

(c)

**Figure 44: Network training**

It is also possible to, with the info obtained as output when training the network, plot the training progress in order to observe how the network is learning. *Figure 45* shows it. In the upper graph it is observed how the network accuracy grows every time a minibatch is introduced to it during the training. In the other graph the loss is plotted, which indicates how bad the network performs.



**Figure 45: Training progress**

Once the network is trained, the test set of images is introduced in order to obtain the test accuracy. As this set of images has not been used before, the results of the crop and weed detection in test images, determines the network's accuracy. One of the outputs of the object detection command (*Figure 46*) is a score, which represents how confident the network is with its predictions. If the score is bigger than a certain index, it means that the object is detected and it will be labelled on the image, if it is not, then it will not be shown. In this thesis, the index has been set to 0.75, therefore, the predictions with less than the 75% of confidence are not shown on the images. After being processed, the final image with all the labels is shown as in *Figure 47*.

```
[dbox, dscore, dlabel] = detect(rcnn, imread(imTest.Files{j}));
```

**Figure 46:** Object detection command



**Figure 47:** Labelled image

When the test images have been labelled, the predictions done to them are evaluated as in *Figure 48*. The average precision is obtained over all the predicted results. By doing this, the performance of the network is tested in images different from those used for its training. Each one of the classes differentiated by the network has its own accuracy. The accuracy obtained with the test images in this thesis is presented in *Table 3*.

```
87 %% Evaluation
88
89 - overlap = 0;
90 - apS = evaluateDetectionPrecision(results, gTruth_Test.LabelData(:,1), overlap);
91 - apD = evaluateDetectionPrecision(results, gTruth_Test.LabelData(:,2), overlap);
92 - apC = evaluateDetectionPrecision(results, gTruth_Test.LabelData(:,3), overlap);
```

**Figure 48:** Test evaluation

| Crop      | Accuracy |
|-----------|----------|
| Spinach   | 77.27%   |
| Cleaver   | 37.54%   |
| Dandelion | 30.08%   |

**Table 3:** Test accuracy

Testing the network is the final step of the network training, but not the final step for this thesis. As the main objective is to differentiate crops and weeds from the FarmBot, once the test accuracy is good enough, a set of prediction images is introduced to the net in order to detect the plants grown in the FarmBot. This set of images is formed by 54 pictures taken by the robot, having photographed the whole FarmBot's bed. To do so, it is necessary to add one last step to the program: plant detection.

### 8.3.3 Plant detection

When it comes to finally detecting the weeds in a set of pictures, the initial steps of the program are similar to the ones done in training and test. Creating a datastore of images, processing them and, if needed, obtaining the ground truth table to later compare the results with reality. In this part, the thesis focuses on the identification of the plants and their localization.

Once each image is correctly processed, the datastore composed by the 54 pictures that conform the FarmBot's field, is put through the detection in order to obtain a prediction of the plants in each image and their corresponding category. To visually represent the predictions, only the labels assigned with more than an 75% of confidence are chosen as valid. This is shown in *Figure 49*.

```
[dbox, dscore, dlabel] = detect(rcnn, imread(imds_pred.Files{j}));  
  
results_pred.Boxes{j} = dbox;  
results_pred.Scores{j} = dscore;  
  
idx = dscore > 0.75;  
dboxTop = dbox(idx, :);  
dlabelTop = dlabel(idx);
```

**Figure 49:** Detection command in Matlab

Visual representation of each image with their correct labels is not the last step, plant detection for this thesis also aims to obtain the coordinates of the existing weeds in each picture, part of the code to do this is represented in *Figure 50*. The program finds, in each image, every label that corresponds to a weed and is considered to be valid. The coordinates of these valid bounding boxes are used to find the centre of each box, representing the location of the crop. Knowing the dimensions of a picture and those of the FarmBot bed, the centre point obtained earlier is transformed to a new coordinate system where the origin is FarmBot's home position: the upper-left corner. Finally, the coordinates of the location of the weeds (cleavers and dandelions) in the FarmBot bed is obtained.

```
if dlabelTop(i) ~= 'Spinach'  
    [coord_x, coord_y] = Coordinates(dboxTop, i, k, j);  
    coord{h} = [coord_x, coord_y];  
    h = h+1;  
end
```

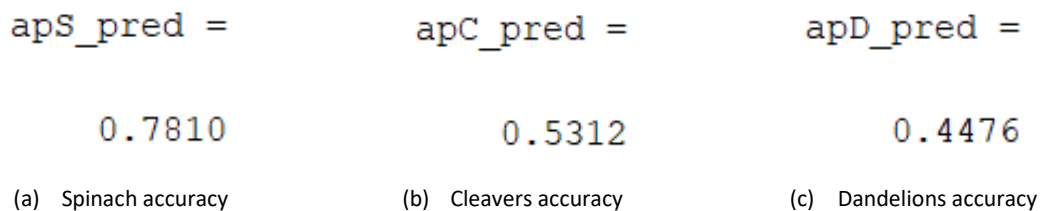
**Figure 50:** *Coordinates obtention in Matlab*

As a brief summary of this part, the Matlab code process the images taken by the robot and trains with them a neural network that performs object detection to the pictures in order to differentiate between spinaches, cleavers and dandelions and, in addition, display the coordinates of both weeds.

## 9. Results

Once the development of the weed detector is finished, the obtained results are: 54 images with the corresponding bounding boxes for each plant detected as well as its label, the precision of the weed detector for each category in this set of images, and a table with the coordinates of the weeds in the FarmBot bed. In addition to these results, a comparison of the performance of the resulting weed detector with the original FarmBot weed detector, as well as one of the neural networks shown in the Literature review (*Section 5.3*), will be made in this chapter.

The most important result of this thesis is the precision obtained from the network. How well does it detect each different type of plant it is presented? These results are shown in *Figure 51 (a), (b) and (c)* for spinach, cleavers and dandelions respectively. These percentages come from all the correct and incorrect guesses the network has made, based on the ground truth table it has been given after the detection. At the beginning of this project a threshold was set to determine what values of accuracy could be considered as a correctly functioning network. This accuracy was set to be of at least 50%. Compared to this initially set value, it can be said that the results obtained are good enough.



**Figure 51:** Detection accuracy

Another result expected from the weed detector is the visualization of the images with their respective bounding boxes surrounding each plant, and their correct label. These visualizations come from a filtering done apart from the neural network, to only show those detections with a confidence above of 75%. In *Figure 52* some examples of this visualization can be found, where it can be seen that the resulting images are not too bad. As seen in *Figure 52 (a)*, the number of plants, their position and their respective label is well determined. In other pictures such as *Figure 52 (b)* it is noticeable how the confidence restriction affects the results, as not all the present plants are detected.

The last result obtained from the developed weed detector is a table containing the location of all detected weed plants in the FarmBot bed. This location is stored as X and Y coordinates in millimeters referred to the home position of the FarmBot: the upper-right corner of the bed. *Figure 53* shows a part of the resulting coordinate table from the images used in this evaluation.





**Figure 52:** Visualization examples

| Weed type | X      | Y      |
|-----------|--------|--------|
| Cleaver   | 1424.8 | 838.92 |
| Dandelion | 1513.2 | 596.41 |
| Dandelion | 1412.2 | 168.09 |
| Cleaver   | 1286.4 | 1263.3 |
| Dandelion | 1316.9 | 338.59 |
| Cleaver   | 1191.6 | 1166.9 |
| Cleaver   | 1151.2 | 982.27 |
| Dandelion | 1105.4 | 622.45 |

**Figure 53:** Coordinate table

After obtaining these results, the main weed detector system and its performance were supposed to be compared to the FarmBot weed detector. A summary of this comparison is done in *Table 4*. The problem is that the FarmBot system does not implement a neural network for the identification but bases the weed detection solely on the crop localization. Due to this, it can work with multiple type of crops and unlimited type of weed, since it does not classify the plants according to their species. It works with only one picture at a time and can return the coordinates of the weeds of every picture. According to FarmBot personnel, their weed detector has an accuracy of approximately 95%. Meanwhile the weed detector developed in this project is based on deep learning for the crop identification and is capable of classifying the plants regardless of their location as long as they are spinach, dandelions or cleavers. It is able of taking 54 pictures at the same time and is also capable of returning the coordinates of the weeds. The accuracy obtained is variable for each plant: 78.10% for spinach, 44.76% for dandelions, and 53.12% for cleavers, as previously said.

Then, if this thesis network accuracy is much lower than FarmBot detection accuracy, why this thesis is stated as successful? It is because a weed detection system is much more intelligent than FarmBot's built-in detector. This means, the resulting neural network from this thesis detects and identifies weeds and crops with less information than FarmBot and learns how to do that identification by its own. FarmBot needs to know where the crops have been planted in advance and its color range, whereas the developed network just needs to be trained with pictures of what would be detected. Therefore, the neural network developed in this thesis is a good result.

| FarmBot weed detector              | Resulting weed detector   |
|------------------------------------|---|
| Based on plant location            | Based on Deep learning  |
| One picture at a time              | 54 pictures at a time   |
| Does not identify the type of weed | Classifies two types of weeds   |
| Multiple crops                     | Only 1 crop: Spinach  |
| Accuracy 95%                       | 78.10% for Spinach, 53.12% for Cleavers,<br>and 44.76% for Dandelions |

**Table 4:** FarmBot weed detector comparison

A comparison with the plant detectors found in the Literature review (*Section 5.3*), is made in order to compare this system with other neural networks. The neural network chosen for this comparison is the one developed by Cordova-Cruzzaty, et al., (2017) due to the similarities in the training setup and the initial network used being Alexnet. This network is a CNN, used for image classification, and not object detection as it has been done with the RCNN for this thesis. While the network in the resulting weed detector proposes regions where there are similarities to the objects that need to be identified, the CNN used in Cordova-Cruzzaty, et al., (2017) uses cropped pictures which only contain one plant to identify. With this method and the training setup shown in *Table 5* Cordova-Cruzzaty, et al., (2017) obtain an accuracy of 92% using nearly 3600 images, while the weed detector developed for this project reports an accuracy of 78.10% for the crops, and 53.12% and 44.76% for each weed type, having used only 227 images for the training.

| Deep Learning Type                       | Crop    | Training Setup  | Training Time                 | Acquisition Setup     | Dataset Strength               | Accuracy % |
|--|---------|---|-------------------------------|-----------------------|--------------------------------|------------|
| Image Based Convolutional Neural Network | Maize   | Core i7 2.7 GHz 8 core CPU Computer with Nvidia GTX950M | Not given                     | Pi camera Version 2.1 | 2835 maize and 880 weed images | 92.08      |
| Object Identification Based RCNN         | Spinach | Intel Core i7 1.99GHz 8 core CPU Computer               | 20 epochs in about 6.30 hours | C925e webcam Logitech | 227 mixed images               | 78.10      |

**Table 5:** Cordova-Cruzzaty network comparison

## 10. Conclusion

In conclusion, the project developed in this thesis has successfully achieved the principal aim set in *Section 1.3*. The principal aim of this thesis was the implementation of a system able to identify crops and weeds using ANNs with images captured by the FarmBot, which will later be compared with FarmBot built-in weed detector. The accuracy obtained is not of a 100%, but the network differentiates well enough the different type of plants it has been trained on. Therefore, the main aim of the thesis is considered as accomplished.

The objectives have been also achieved. The FarmBot has been used to take the pictures and then those pictures have been correctly processed by Matlab, an ANN was trained and tested and then used to predict the crop and weeds at the pictures and also return the weed coordinates. Finally, a comparison with other methods was done. Therefore, all the objectives have succeeded.

Through the awareness step of the methodology, enough information was acquired to propose an initial design of the project. This prototype gives an initial idea of how to obtain the images with FarmBot and Python, and as well as a possible approach to the network training and testing. Once proved feasible, it was evaluated, and some modifications were proposed. The main changes implemented in the final version were the addition of image processing, the change from image classification to object identification by changing the type of neural network from CNN to RCNN in the training part, and the addition of the coordinate obtention as the final part after the plant identification.

To evaluate the resulting system after the development, the weed detector was tested as it is supposed to work in a real situation. First the FarmBot programme was tested to obtain a whole set of pictures from the FarmBot bed. Then the Matlab detection programme was executed, firstly making the python code execute the image acquisition and then detecting the contents of each image. The results of this evaluation can be found in *Chapter 9* where the performance, accuracy and coordinates obtained are shown, as well as a comparison of the network with the original FarmBot weed detector and another similar network found during the research step.

These results prove the correct development of the project, as the aim and objectives have been successfully achieved as mentioned above. The developed system takes care of additional features the FarmBot weed detector does not consider such as not basing the detection only on known locations, or not identifying the type of weeds. Moreover, with inferior training setups and dataset strength, the developed system is still able to obtain satisfactory accuracy percentages. This proves that, even if the accuracy obtained is smaller than the accuracy of the compared systems, the resulting weed detector meets the expectations, although leaving room for improvement.

## 11. Discussion

The detection accuracy results achieved in this project are not as high as in other projects with object identification networks, nevertheless, they are good enough to lead to the implementation of artificial intelligence in a new field: agricultural robots such as FarmBot. Crop recognition has been used before in the agricultural field, but focusing on FarmBot, the project developed in this thesis entails an improvement to the existing weed detector system, in spite of the accuracy obtained.

Despite having achieved the objectives set for this thesis, there is still room for improvement. The main issue that came up during the development was the poor camera characteristics, as the cameras used were too sensitive to light. The use of another camera as well as using polarizers over its lens would be possible improvements to consider. Changing the environment of the picture taking to reduce direct lighting over the FarmBot bed would also help avoid unwanted glare. This will most certainly improve the quality of the images, thus improving the network's performance.

### Future work

Here, some possible future projects based on this thesis are proposed. A deeper research on the topic of this thesis can be done by considering the creation of the neural network from scratch, instead of using transfer learning on an existing network. By creating the network architecture there can be more control over its learning process. Another possible topic to research based on this thesis is the implementation of a weed detector taking into account a bigger number of crop and weed types, not only spinach, cleavers and dandelions. Finally, the implementation of an automatic FarmBot weed removal based on deep learning detection would be an interesting topic to consider, given that the current weed removal system of FarmBot cannot detect weeds located near a crop.

## References

- Alpaydin, E., 2016. *Machine Learning: The New AI*. Cambridge: The MIT Press.
- Alves Varella, C. A., Marinaldo Gleriani, J. & Medeiros dos Santos, R., 2015. Precision Agriculture and Remote Sensing. In: F. Santos, A. Borém & C. Caldas, eds. *Sugarcane: Agricultural Production, Bioenergy and Ethanol*. s.l.:Academic Press, pp. 185-203.
- Angeles, J., 2014. *Fundamentals of Robotic Mechanical Systems*. Fourth ed. Switzerland: Springer.
- Bah, M. D., Hafiane, A. & Canals, R., 2017. Weeds detection in UAV imagery using SLIC and the hough transform. *2017 Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pp. 1-6.
- Bansal, S., 2017. *Supervised and Unsupervised learning*. [Online]  
Available at: <https://www.geeksforgeeks.org/supervised-unsupervised-learning/>  
[Accessed 02 03 2020].
- Brown, J. et al., 2017. *Polar Coordinate FarmBot Final Project Report*, California Polytechnic State University: s.n.
- Brundtland, G. H., 1987. *Report of the World Commission on Environment and Development: Our Common Future*. Oslo, s.n.
- Chavan, R., T. & Nandedkar, A. V., 2018. AgroAVNET for crops and weeds classification: A step forward in automatic farming. *Computers and Electronics in Agriculture*, Issue 154, pp. 361-372.
- Córdova-Cruzatty, A. et al., 2017. *Precise Weed and Maize Classification through Convolutional Neural Networks*, Sangolquí, Ecuador: s.n.
- Daman, M., Aravind, R. & Kariyappa, B., 2015. Design and Development of Automatic Weed Detection and Smart Herbicide Sprayer Robot. *IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, pp. 257-261.
- Dankhara, F., Patel, K. & Doshi, N., 2019. *Analysis of robust weed detection techniques based on the Internet of Things (IoT)*. Coimbra, Portugal, Elsevier B.V, pp. 696-701.
- De Baerdemaeker, J., 2013. Precision Agriculture Technology and Robotics for Good Agricultural Practices. *IFAC Proceedings Volumes*, 46(4), pp. 1-4.
- Deng, L. & Yu, D., 2014. Deep Learning: Methods and Applications. *Foundations and Trends in Signal Processing*, 7(4), pp. 197-387.
- Dumka, A. & Sha, A., 2019. Smart ambulance system using concept of big data and internet of things. In: N. Dey, C. Bhatt, A. S. Ashour & S. J. Fong, eds. *Healthcare Data Analytics and Management*. s.l.:Academic Press, pp. 155-176.

Ertel, W., 2017. *Introduction to Artificial Intelligence*. Second ed. Weingarten, Germany: Springer.

FarmBot Inc., 2018. *FarmBot Inc.* [Online]  
Available at: <https://farm.bot/pages/series-a>  
[Accessed 18 02 2020].

Fawakherji, M., Youssef, A. B. D. D. P. A. & Nardi, D., 2019. *Crops and Weeds Classification for Precision Agriculture using context-independent Pixel-Wise Segmentation*. s.l., s.n., pp. 146-155.

Gupta, A. & Gupta, P., 2015. *Electrical Energy Efficient System Design using Automation and Artificial Intelligence for Classrooms*, s.l.: s.n.

Irías Tejeda, A. J. & Castro Castro, R., 2019. *Algorithm of Weed Detection in Crops by Computational Vision*. Cholula, Mexico, IEEE.

Jähne, B. & Haussecker, H., 2000. Introduction. In: *Computer Vision and Applications*. San Diego: Academic Press, pp. 1-8.

Jha, K., Doshi, A., Patel, P. & Shah, M., 2019. A comprehensive review on automation in agriculture using artificial intelligence. *Artificial Intelligence in Agriculture*, pp. 1-12.

Knoll, F. J. et al., 2018. Improving efficiency of organic farming by using a deep learning classification approach. *Computers and Electronics in Agriculture*, pp. 347-356.

Kok, J. N., Boers, E. J. W., Kusters, W. A. & van der Putten, P., 2009. *Artificial Intelligence: Definition, trends, techniques, and cases*. s.l.:EOLSS.

Kurry, A., 2011. *Sustainable Development*. [Online]  
Available at: <https://eportfolios.macaulay.cuny.edu/akurry/2011/12/21/sustainable-development/>  
[Accessed 18 02 2020].

Liang, W.-C., Yang, Y.-J. & Chao, C.-M., 2019. Low-Cost Weed Identification System Using Drones. *Candarw*, Volume 1, pp. 260-263.

Lu, Y. & Lu, R., 2016. Quality Evaluation of Apples. In: D. Sun, ed. *Computer Vision Technology for Food Quality Evaluation*. s.l.:Academic Press, pp. 273-304.

Marinoudi, V., Sorensen, C., Pearson, S. & Bochtis, D., 2019. Robotics and labour in agriculture. A context consideration. *Biosystems Engineering*, pp. 111-121.

Marzuki Mustafa, M., Hussain, A., Hawari Ghazali, K. & Riyadi, S., 2007. *Implementation of Image Processing Technique in Real Time Vision System for Automatic Weeding Strategy*. Bangi, Malaysia, IEEE.

Mataric, M. J., 2007. What Is a Robot?. In: *The Robotics Primer*. Massachusetts: The MIT Press, pp. 1-6.



- McAndrew, A., 2004. *An introduction to Digital Processing with Matlab. Notes for scm2511 image processing..* School of Computer Science and Mathematics, Victoria University of technology, 264(1): s.n.
- McClelland, C., 2020. *What is IoT? - A Simple Explanation of the Internet of Things.* [Online] Available at: <https://www.iotforall.com/what-is-iot-simple-explanation/> [Accessed 14 02 2020].
- McCool, C., Perez, T. & Upcroft, B., 2017. Mixtures of Lightweight Deep Convolutional Neural Networks: applied to agricultural robotics. *IEEE Robotics and Automation Letters*, pp. 1-8.
- Mehta, P., 2016. Automation in Agriculture: Agribot the Next Generation Weed Detection and Herbicide Sprayer - A Review. *Journal of Basic and Applied Engineering Research*, 3(3), pp. 234-238.
- Miloto, A., Lottes, P. & Stachniss, C., 2017. *Real-Time Blob-Wise Sugar Beets vs Weeds Classification For Monitoring Fields Using Convolutional Neural Networks.* Bonn, Germany, International Conference on Unmanned Aerial Vehicles in Geomatics.
- Moazzam, S. I. et al., 2019. *A Review of Application of Deep Learning for Weeds and Crops Classification in Agriculture.* Rawalpindi, Pakistan, s.n., pp. 1-6.
- Neapolita, R. E. & Jiang, X., 2018. *Artificial Intelligence with an introduction to Machine Learning.* Second ed. s.l.:CRC Press.
- Niku, S. B., 2020. *Introduction to Robotics: Analysis, Control, Applications.* Third ed. California: Wiley.
- Nixon, M. & Aguado, A., 2019. *Feature Extraction and Image Processing for Computer Vision.* s.l.:Academic Press.
- Nouzil, I., Raza, A. & Pervaiz, S., 2017. *Social aspects of automation: Some critical insights..* Dubai, IOP Publishing.
- Oates, J. B., 2006. *Researching Information Systems and Computing.* London: Sage Publications.
- Olsen, A. et al., 2015. *In Situ Leaf Classification Using Histograms of Oriented Gradients.* Adelaide, SA, Australia, s.n.
- Pantazi, X. E., Dimitrios, M. & Bochtis, D., 2020. Chapter 1- Sensors in agriculture. In: *Intelligent Data Mining and Fusion Systems in Agriculture.* s.l.:Academic Press, pp. 1-15.
- Pedersen, S. M., Fountas, S., Have, H. & Blackmore, B. S., 2006. Agricultural robots - system analysis and economic feasibility. *Precision Agric*, Volume 7, pp. 295-308.
- Pramanik, P. K. D., Upadhyaya, B. K. & Pal, T., 2019. Internet of things, smart sensors, and pervasive systems: Enabling connected and pervasive healthcare. In: N. Dey, C. Ghatt, A. S. Ashour & S. J. Fong, eds. *Healthcare Data Analytis and Management.* s.l.:Academic Press, pp. 1-58.



Reina Terol, A. J., 2019. *Campus Virtual UMA*. [Online]  
Available at: <https://eii.cv.uma.es/course/view.php?id=886>  
[Accessed 02 03 2020].

Samarajeewa, T., 2013. *Identification of Lantana Camara Distribution Using Convolutional Neural Networks*, University of California Santa Cruz: s.n.

SAS, 2019. *Computer vision: what is it and why it matters*. [Online]  
Available at: [https://www.sas.com/en\\_us/insights/analytics/computer-vision.html](https://www.sas.com/en_us/insights/analytics/computer-vision.html)  
[Accessed 15 02 2020].

Sharma, O., 2019. Deep Challenges Associated with Deep Learning. In: *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. Faridabad, India: s.n., pp. 72-75.

Shukla, S., 2017. *Regression and Classification / Supervised Machine Learning*. [Online]  
Available at: <https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/>  
[Accessed 02 03 2020].

Skansi, S., 2018. *Introuction to Deep Learning: From logical calculus to artificial intelligence*. Zagreb, Croatia: Springer.

Tang, J. et al., 2017. Weed identification based on K-means feature learning combined with convolutional network. *Computers and Electronics in Agriculture*, Issue 135, pp. 63-70.

United Nations, 2020. *Sustainable Development GOALS*. [Online]  
Available at: <https://www.un.org/sustainabledevelopment/development-agenda/>  
[Accessed 18 02 2020].

Vamshidhar Reddy, N., Vishnu Vardhan Reddy, A. V., Pranavadithya, S. & Jagadesh Kumar, J., 2016. A Critical Review on Agricultural Robots. *International Journal of Mechanical Engineering and Technology (IJMET)*, 7(4), pp. 183-188.

Wafy, M., Ibrahim, H. & Kamel, E., 2013. Identification of weed seeds species in mixed sample with wheat grains using SIFT algorithm. *2013 9th International Computer Engineering Conference (ICENCO)*, pp. 11-14.

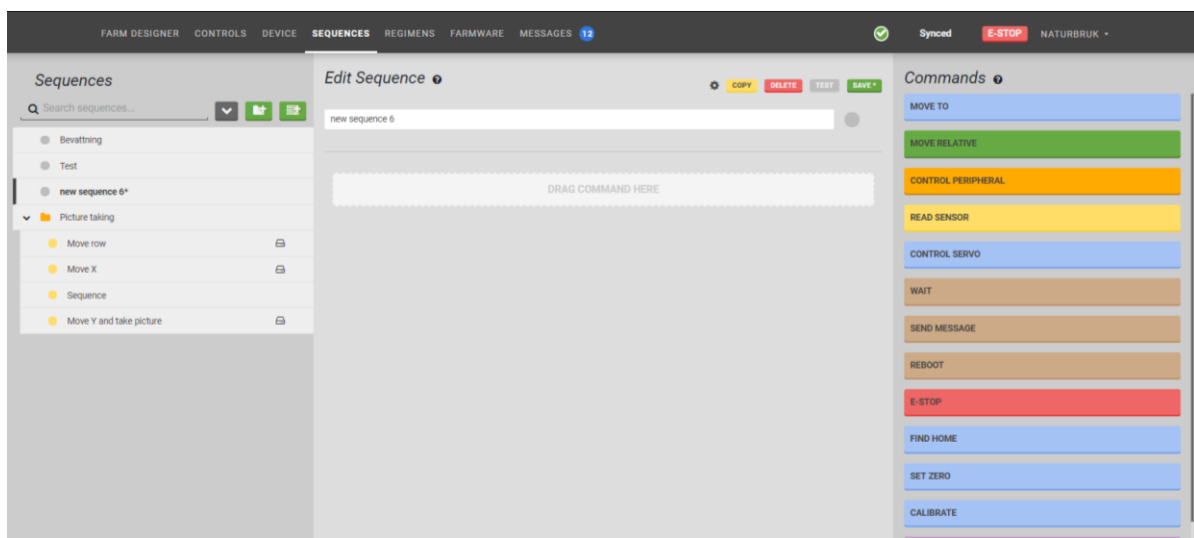
Wason, R., 2018. Deep Learning: Evolution and expansion. *Cognitive Systems Research* , Issue 52, pp. 701-708.

Worldometers, 2020. *World population*. [Online]  
Available at: <https://www.worldometers.info/world-population/>  
[Accessed 02 02 2020].

## Appendix A: Project prototype programming

### 1. FarmBot

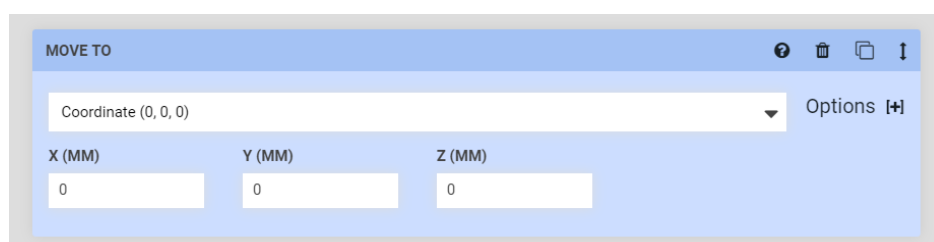
To guarantee the best accuracy possible, the CNN needs to be trained and tested with images taken from the FarmBot. A code where these images are taken has been developed in the FarmBot UI, an application from where the user can control the robot remotely. The FarmBot UI is shown in *Figure 54*.



**Figure 54:** FarmBot UI

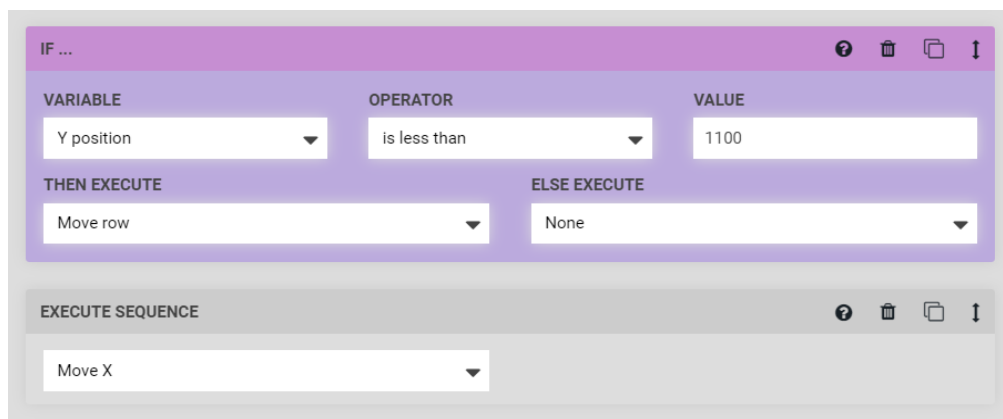
Not only does the robot have to take the pictures but also moves to the correct place in order to take photos of all the crops. The robot's workspace can be represented with 25 pictures, organised in five rows (in the Y axis) and five columns (in the X axis). To do so, different sequences have been programmed: 'Move row', 'Move X', 'Sequence' and 'Move Y and take picture'. The main code is 'Sequence'. All the other sequences used in this program are called from here which is why, to run the full code, it is only necessary to run 'Sequence'.

The first step is shown in *Figure 55*, it moves the robot to a safe position, which in this case is the origin of the coordinate system {0, 0, 0}. By having this position, it is ensured that the pictures will be taken in the desired order to later be processed.



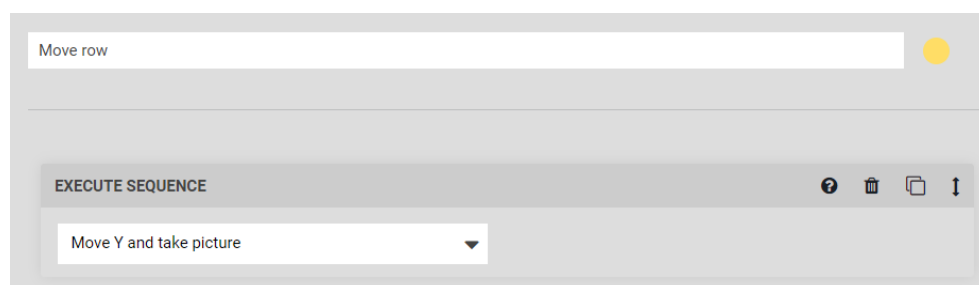
**Figure 55:** Move to origin

Once it is in its initial position, it is ready to take pictures. The next two commands, which are shown in *Figure 56*, make the robot take pictures of a full row. This is made possible by executing the 'Move row' sequence until the position of the robot reaches 1.10 meters in the Y axis, which would mean that it has reached the end of the workspace in that axis. After having moved through the whole row, it executes 'Move X'. This is done four times since the workspace has five rows; the final row is done differently and will be explained later. Both sequences 'Move row' and 'Move X' will be explained below.

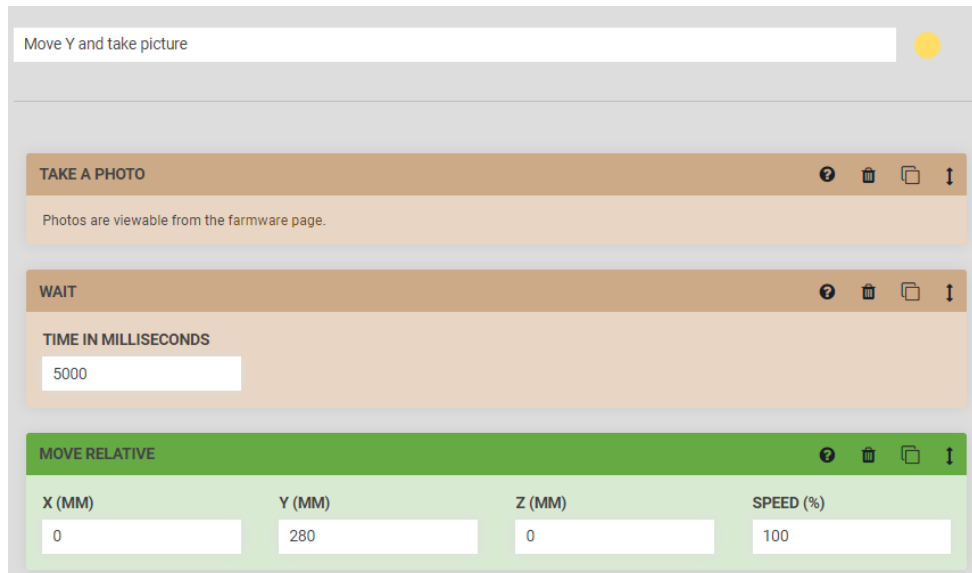


**Figure 56:** Take pictures of a complete row

'Move row' (*Figure 57*) as its own name says, moves the robot along the rows. This is done by executing the sequence 'Move Y and take picture' (*Figure 58*) four times. It works as follows: once the robot is in its home position, it takes a picture, waits 5 seconds to ensure the picture was correctly taken and then moves 0.28 meters in the Y axis in order to take the following picture. It is exactly 0.28 meters because it is the length of Y axis divided into the five pictures that will be taken there. Once four of those five pictures are taken, the robot's Y position will be greater than 1100 so it will exit the IF statement in *Figure 66* and execute 'Move X'.

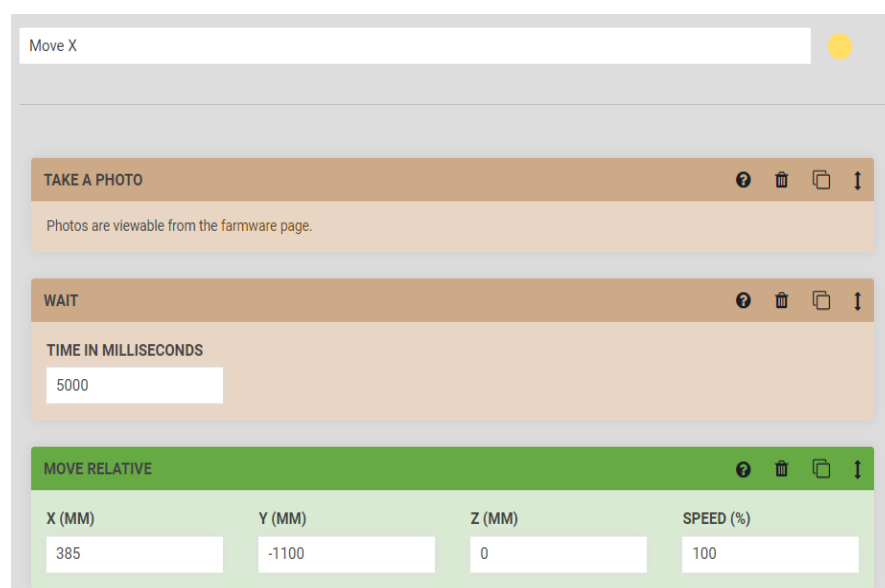


**Figure 57:** Move row sequence



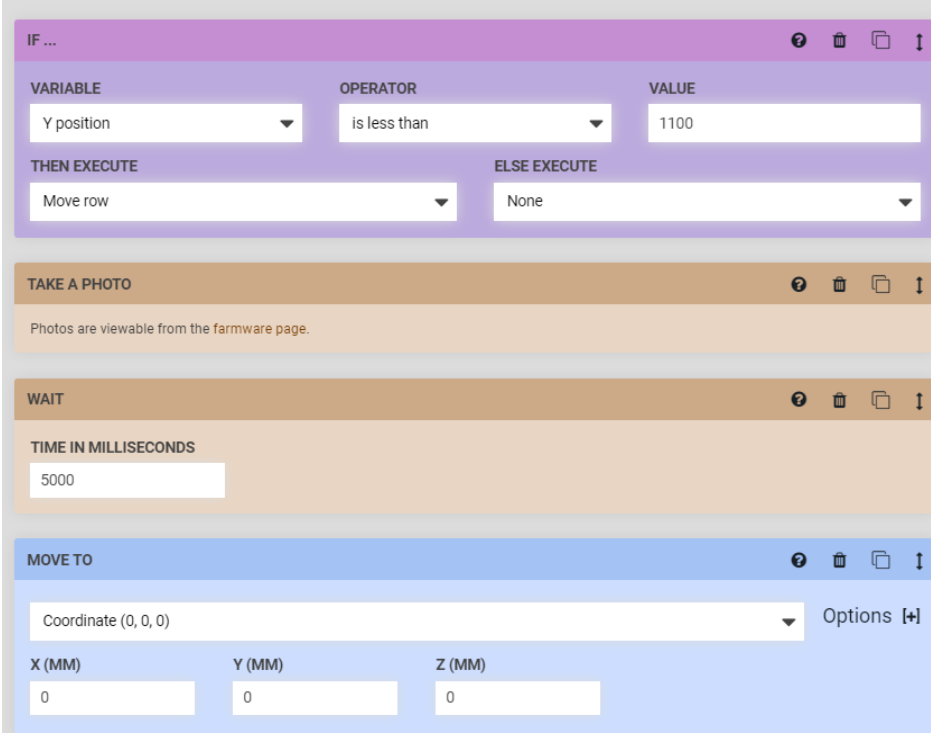
**Figure 58:** 'Move Y and take picture' sequence

As said before, only four out five pictures in the row are taken in 'Move row', the last picture is taken in 'Move X' sequence as shown in *Figure 59*. This is done because in the sequence above, moving is the last step but in the fifth picture it won't be able to move because it is at the end of the axis; then, the picture will be taken before changing column instead of row. Therefore, 'Move X' takes a photo, waits 5 seconds again to ensure it is correctly taken and then moves back to the first column (moving 1.1 meters in the opposite direction of the Y axis) and to the next row (0.385 meters in the X axis, also calculated taking into account the number of pictures and the length of the axis).



**Figure 59:** 'Move X' sequence

After having executed the IF statement and the 'Move X' commands four times, the IF command is done one last time in order to take the pictures of the last row and, instead of executing 'Move X', the robot takes one last picture, waits 5 seconds again and finally move to the {0, 0, 0} location, as shown in *Figure 60*.



The screenshot displays a sequence of four commands in a FarmBot interface:

- IF ...** (Purple header):
  - VARIABLE: Y position
  - OPERATOR: is less than
  - VALUE: 1100
  - THEN EXECUTE: Move row
  - ELSE EXECUTE: None
- TAKE A PHOTO** (Orange header):
  - Photos are viewable from the farmware page.
- WAIT** (Orange header):
  - TIME IN MILLISECONDS: 5000
- MOVE TO** (Blue header):
  - Coordinate (0, 0, 0)
  - X (MM): 0
  - Y (MM): 0
  - Z (MM): 0

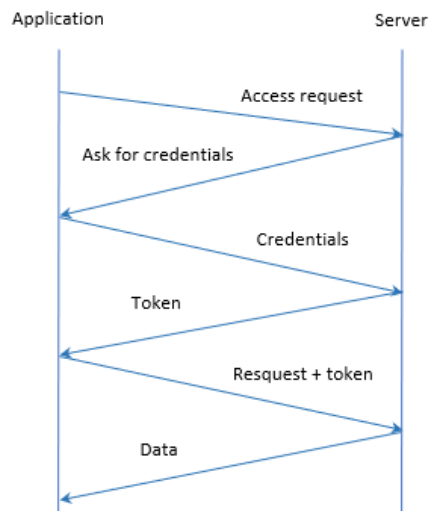
**Figure 60:** Final commands

## 2. Python

The pictures taken by the FarmBot are stored in its Google Cloud Storage; as the access to the cloud is not given, in order to download them it is necessary to ask for it, and this is done by sending a request through the FarmBot API. To do so, a code in Python language able to send the request to the API and download the pictures has been developed.

An API is an interface where the way applications interact and exchange data between them is specified. It provides the platform, medium and specifications about how the information is structured in each application in order to make possible the communication among various applications. One type of structure is REST API, it specifies how data is represented in a format convenient for the client, including its security, being data exchanged mostly in JSON format. FarmBot uses REST API as its architecture to control storage, image upload and manipulation and security, and JSON Web Token as authorization mechanism.

An API token is a way to relate an application requesting access to a service to a unique identifier. The first time an application requests access to your service, some credentials are asked for; if they are valid, an API token would be generated; that token will be used as a form of authentication, the client stores that token and sends it with every request so the server verifies it and responds with the data requested; this process is shown in *Figure 61*.



**Figure 61:** Data acquisition using API tokens

The Python code written for this thesis to access the FarmBot pictures follows the same flow shown in *Figure 61*, and it is explained in detail bellow. In this thesis, the application is the code and the server is the FarmBot web application.

First of all, all the required libraries are imported in order to being able to use the proper commands (*Figure 62*).

```
1  import requests
2  import os
3  import json
4
```

**Figure 62:** Libraries imported

Once the libraries are ready, the token is asked for. It is done as shown in *Figure 63*. The token asked for is stored in a variable called 'response' in order to save the access and being able to connect again. For the generation of the token it is necessary to send the server certain information: a method, the server URL, some headers and the credentials. In this case the method will be 'POST' because a new resource or connection is being created, the URL is FarmBot web application's internet link, the

headers are where the expected type of content is determined, and finally, the credentials, sent as a JSON object, are the username and the password needed to access the correct FarmBot server. The token received as answer is stored in the variable 'TOKEN' encoded as a JSON.

```
6
7 # Generating token
8 response = requests.request(
9     method='POST',
10    url='https://my.farm.bot/api/tokens',
11    headers={'content-type': 'application/json'},
12    json={'user': {'email': '*****', 'password': '*****'}})
13
14 TOKEN = response.json()['token']['encoded']
15
16
```

**Figure 63:** Token generation code

After receiving the proper token, the next step is accessing the images stored by FarmBot as *Figure 64* shows. As an authorization is required to do so, the token and the type of content expected for the answer are introduced in a variable called 'headers'; a request is then sent to get the images by sending both 'headers' and the URL to be accessed. The response to this request is returned as a JSON object that contains diverse data related to the images such as the creation date, name, URL, and even the camera coordinates where the pictures were taken, among other data. This information is stored as a new variable named 'images'.

```
16
17 # Get images url
18 headers = {'Authorization': TOKEN,
19            'content-type': "application/json"}
20 response = requests.get('https://my.farmbot.io/api/images', headers=headers)
21 # print(response)
22 images = response.json()
23
```

**Figure 64:** Accessing the images

Having accessed the images, it is now possible to download them to the local storage. The first step is to determine the number of available images. For every image, its URL and camera coordinates are stored separately; then the image is downloaded from the URL and saved with a unique name in a previously specified path, lastly, the same is done with its camera coordinates. The code representing this process is shown in *Figure 65* and *Figure 66*.

```
4
5 path = 'C:/Users/User/Desktop/TFG/Matlab/'
6
```

**Figure 65:** Path determination

```

24 # Number of images
25 numel = len(images)
26
27 # Download data and images
28 for i in range (0, numel-1):           # From the latest picture (0) to the oldest (numel-1)
29     image_url = images[i]['attachment_url']
30     image_data = images[i]['meta']
31     print(image_url)
32     print(image_data)
33     myfile = requests.get(image_url)   # Download the picture
34     open(path + 'im' + str(i+1) + '.jpg', 'wb').write(myfile.content) # Save the picture
35     open(path + 'data' + str(i+1) + '.txt', 'wt').write(json.dumps(image_data)) # Save the picture data

```

**Figure 66:** Image download

### 3. Matlab

An important point of the development of this project is the correct classification between crops and weeds. A program is developed in Matlab to modify, train, validate and, lastly, test a CNN, with the goal of it being able to correctly classify the images it is presented. As this is the prototype of the main program, it is implied that it is not the final version, and as modifications will occur, the final decisions as well as the changes made are explained accordingly in *Chapter 8* of the report.

The first step is to access the images needed. As explained in previous chapters, the images obtained from the acquisition process are now available in the chosen path. In order to work comfortably with these pictures, an imageDatastore is created, as shown in *Figure 67*. In Matlab, an ImageDatastore object is a variable capable of accessing image files from a specific path stored in a computer, without importing the data directly into the program. With this variable, it is possible to manage a large collection of images instead of working with the individual files, without worrying about the memory management. It is also possible to access properties such as the labels of each image, which determines whether they are crops or weeds, based on the folder of their location; this property is fundamental for the training step of this process. In addition to this, as this type of object is also a valid input for the CNN, it is a good option for the image management of this project.

```

1 %% Create data store:
2
3 - path = '\\studentnas1\studenthome\2019\b191auma\Desktop\TFG\Pictures';
4 - imds = imageDatastore(path, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
5

```

**Figure 67:** Datastore creation

Once the images have been accessed and the Datastore has been created, it is necessary to randomly split the images in three lots: one for the training of the CNN, one for its validation and another for the testing. Each of these steps of the programming require a unique set of images to ensure the network does not memorize the given classifications during the training step, and that it



correctly learns to differentiate new images. *Figure 68* shows how each new dataset is augmented, modifying some properties of the images such as their size and their colour. The image dimension must be the same as the input dimensions for the CNN chosen.

```
6  %% Split training, test and validation:
7
8  [trainImgs, valiImgs, testImgs] = splitEachLabel(imds, 0.6, 0.2, 'randomized');
9  trainInds = augmentedImageDatastore([227 227], trainImgs, 'ColorPreprocessing', 'rgb2gray');
10 valids = augmentedImageDatastore([227 227], valiImgs, 'ColorPreprocessing', 'rgb2gray');
11 testInds = augmentedImageDatastore([227 227], testImgs, 'ColorPreprocessing', 'rgb2gray');
12
```

**Figure 68:** Splitting and augmenting the image datastore

AlexNet is the CNN used during this project, and it takes an image input of dimensions 227-by-227-pixels. This convolutional network has been pre-trained on more than a million images and can now classify into 1000 classes. AlexNet needs to be taught to classify the specific images in this project with their correct labels, which are not included in its known categories. The method chosen, in order to teach this network to classify between the crops and weeds of this project, is transfer learning. Transfer Learning consists on the modification of the layers of a pre-trained network, in this case AlexNet, and the training of these layers with a set of known labelled images, the training datastore, with a specific algorithm.

```
13  %% Net modification:
14
15  net = alexnet;
16  layers = net.Layers;
17  layers(23) = fullyConnectedLayer(2);
18  layers(25) = classificationLayer;
19  opts = trainingOptions('sgdm', 'InitialLearnRate', 0.001, 'ValidationData', valids);
20  [trainedNet, info] = trainNetwork(trainInds, layers, opts);
```

**Figure 69:** Network modification and training options

As shown in *Figure 69*, this step starts by loading the pre-trained network into a variable called `net`. Examining the layers of this network, the ones that need to be modified are the last fully connected layer and the classification layer. The fully connected layer is where the number of classes is determined; in AlexNet this layer has 1000 neurons representing the original 1000 categories it was trained on. In this project, only two classes are needed: weed and crop. The classification layer is where the output of the network is given. To change the layers accordingly, these modifications are done as shown in lines 17 and 18 of *Figure 69*.

The training algorithm is specified in line 19 of *Figure 69*. The first value of the training options is 'SGDM', which stands for Stochastic Gradient Descent with Momentum, it is an algorithm for the learning approach. Another relevant parameter is the Initial Learn Rate, this value determines the range in which the network's parameters should be modified during the learning. The third and final parameter introduced in this code as a training option is the validation datastore, it allows the training process to verify the progress of the training validating it with a third set of images and labels. These settings are the training options. In line 20 of *Figure 69*, the network is trained with these specifications, returning the trained network and information about accuracy.

The final steps, after the network training, are the image classification and the network's performance evaluation. The command 'classify' takes the trained network and the test datastore as the inputs, returning the prediction for each image. The evaluation of the network is then done by obtaining the percentage of correctly predicted images in line 25 of *Figure 70*, and lastly, a confusion chart is drawn with the results obtained from the classification, to visually determine which predictions were wrong.

```
22 %% Classification y evaluation
23
24 - Preds = classify(trainedNet,testds);
25 - accu = nnz(Preds == testImgs.Labels)/numel(Preds);
26 - confusionchart(testImgs.Labels, Preds)
```

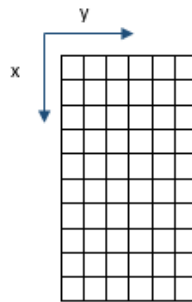
**Figure 70:** Evaluation

## Appendix B: Final project code

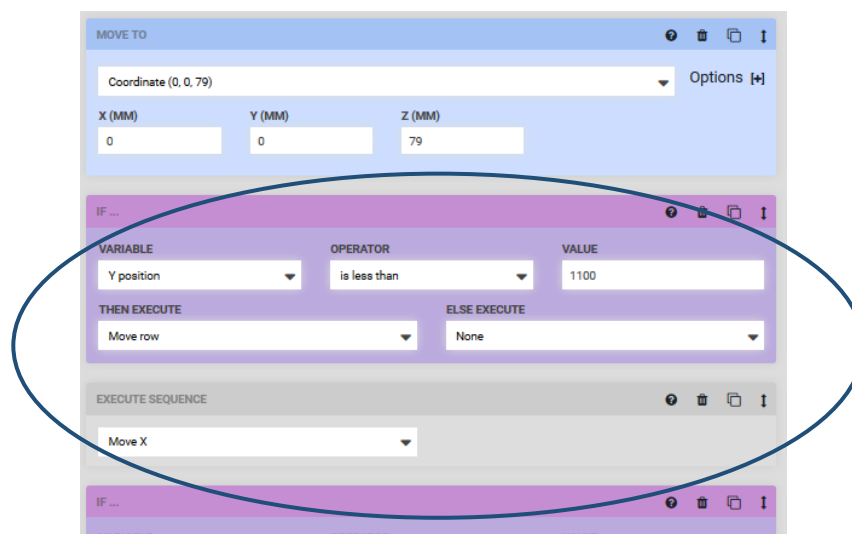
### 1. FarmBot

The code developed for this project prototype has not many changes. The principal modification done is the number of pictures taken. As the amount of photos has varied, the number of times the sequences are executed has also been modified.

In *Section 8.1* has been explained that, due to the measures of the FarmBot and its pictures, 54 pictures are going to be taken in order to process the whole FarmBot's bed which can be measured as 9x6 pictures (nine in its X axis and six in the Y one) as shown in *Figure 71*. This leads to a higher number of movements needed. The full code developed for the image acquisition is shown in the following figures, being divided into four parts as before, the main sequence (*Figures 72 and 76*), 'Move row' (*Figure 73*), 'Move Y and take picture' (*Figure 74*) and 'Move X' (*Figure 75*). They will not be explained in detail since the way the work has not changed from the code explained in *Appendix A*, only the bigger modifications will be highlighted.

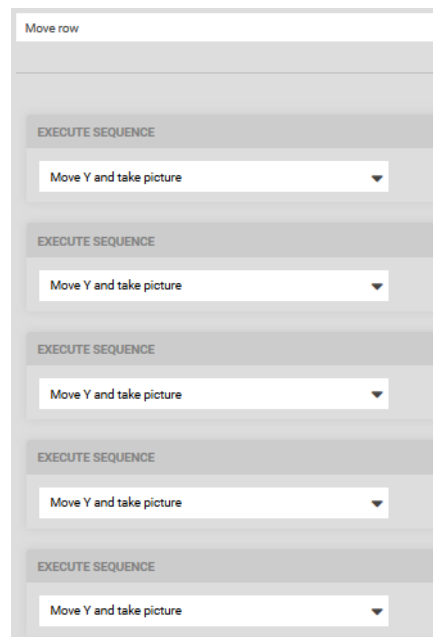


**Figure 71:** FarmBot's picture division

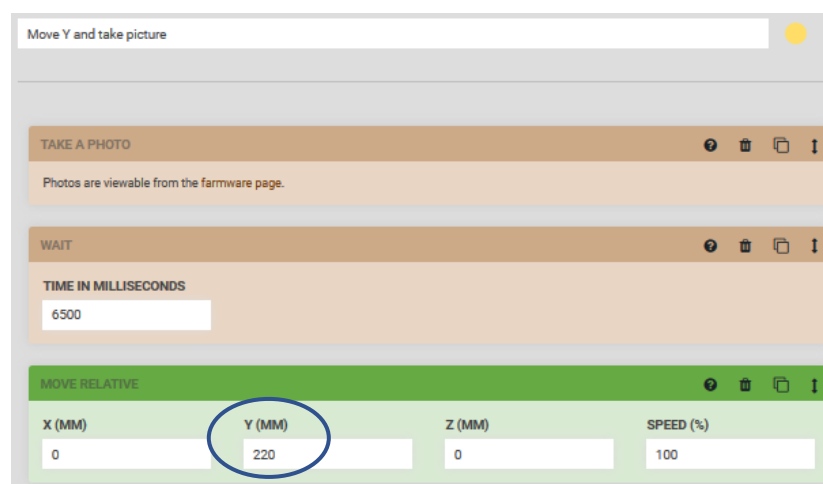


**Figure 72:** Final main sequence (I)

As in the prototype, the robot moves home and then it starts moving row by row as if the field was a matrix where the rows are as wide as a picture. The main difference now is that the circled commands are going to be repeated eight times in order to take pictures of the FarmBot's X axis completely except for the last row which is slightly different. Therefore, 'Move row' (Figure 73) and 'Move X' (Figure 74) sequences are repeated eight times.



**Figure 73:** Final 'Move row' sequence

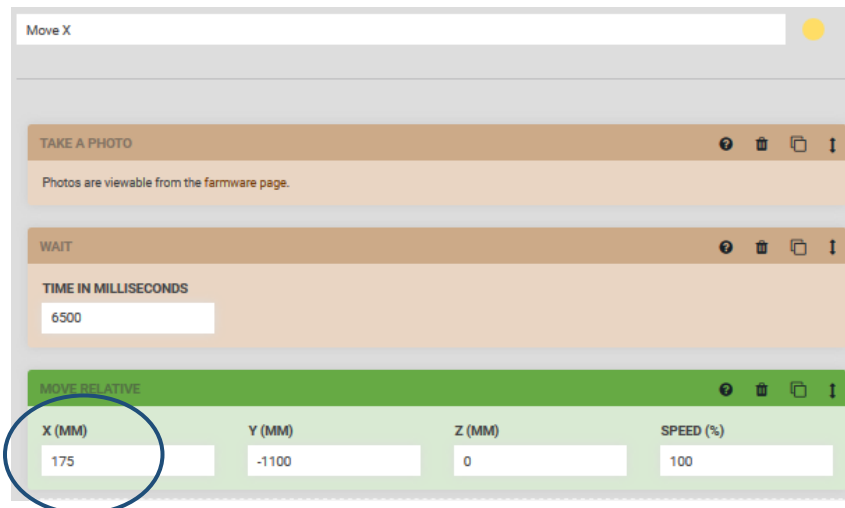


**Figure 74:** Final 'Move Y and take picture' command

In order to take pictures of the whole Y axis, the sequence 'Move Y and take picture' is executed five times until it achieves the last position where the picture has to be taken and then the robot moves to the next row. One modification to take into account in the 'Move Y and take picture command' from

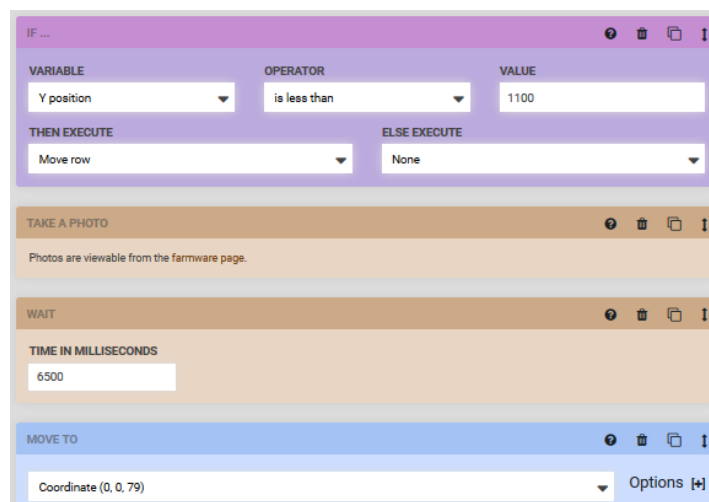
the prototype can be seen in *Figure 74* and is the distance the robot moves to take the next picture, 220 mm now, which is the picture width.

Once the Y axis has been photographed, it takes one last picture and moves to the beginning of the next row, which is 175 mm behind in the X axis. This is another modification from the prototype because of the pictures size.



**Figure 75:** Final 'Move x' command

Finally, when the robot is at its last position, meaning last row, last column; it takes the last picture and moves home. Then it is ready to start working again. One last modification that is shown in every picture in this chapter is the amount of time the robot waits to move after having taken a picture. In the prototype it was five seconds, but now it is six seconds and a half, in order to make sure the picture is taken with as much quality as possible.



**Figure 76:** Final main sequence (II)

## 2. Python

The full python code represented in *Figure 77* is almost identical to the one developed in the initial design. As explained in *Chapter 8*, only two changes have been made for the final version: the removal of unnecessary pictures and the number of images downloaded from FarmBot.

The first change is implemented with python as shown in lines 22 to 26 of the image below, all previous FarmBot images stored in the computer are deleted in order to make room for the new pictures. The second change is shown in line 29 of the code, where the range of 'i' is set to go from 0 to 53. This way, the program will only download the 54 latest pictures taken with FarmBot, which represent the whole length and width of the FarmBot bed where the crops are planted.

```
1  import requests
2  import os
3  import json
4
5  path = 'C:/Users/User/Desktop/TFG/Matlab/FarmBot/'
6
7  # Generating token
8  response = requests.request(
9      method='POST',
10     url='https://my.farm.bot/api/tokens',
11     headers={'content-type': 'application/json'},
12     json={'user': {'email': 'farmbot@naturbruk.nu', 'password': 'Sotasen2018'}})
13
14  TOKEN = response.json()['token']['encoded']
15
16  # Get images url
17  headers = {'Authorization': TOKEN,
18            'content-type': "application/json"}
19  response = requests.get('https://my.farmbot.io/api/images', headers=headers)
20  images = response.json()
21
22  #Deleting previous images in computer
23  files = 'C:/Users/User/Desktop/TFG/Matlab/FarmBot'
24  filesToRemove = [os.path.join(files,f) for f in os.listdir(files)]
25  for f in filesToRemove:
26      os.remove(f)
27
28  # Download data and images
29  for i in range(0, 53): # From the latest picture (0) to picture number 54
30      image_url = images[i]['attachment_url']
31      image_data = images[i]['meta']
32      myfile = requests.get(image_url) # Download the picture
33      open(path + 'im' + str(i+1) + '.jpg', 'wb').write(myfile.content) # Save the picture
34      open(path + 'data' + str(i+1) + '.txt', 'wt').write(json.dumps(image_data)) # Save the picture data
```

**Figure 77:** Python image acquisition

### 3. Matlab

Matlab programming is the main part of the thesis, as the full code is too long to be explained in the development of the project, it can be found bellow. It has been divided in three parts according to its function: image processing, network training and plant detection.

Before doing any processing, Matlab needs to access the pictures downloaded with the Python code. In order to make that code work, it is called from Matlab by using the command in *Figure 78*. Then, once the images are downloaded, the image processing step is next.

```
pyversion  
system('Image_download.py')
```

**Figure 78:** Execute Python command in Matlab

#### 3.1 Image processing

In the Matlab program the pictures are firstly loaded in a datastore as it is done in *Figure 79*. Then, as shown in *Figure 80*, the next step is image processing to improve the picture's condition. The image processing is done to each and every picture contained in the datastore, which will be later used either for training or testing.

As explained in *Chapter 7*, with this excerpt of code the bright white lights caused by overexposure are reduced by giving new values to the bright pixels, taking into account the neighboring pixel values. Sharpening is also applied to the pixels of the image, in order to reduce the blurring and emphasizing the edges of different objects. This helps the distinction between soil and plants. Finally, once the processing of an image has been completed, the picture is saved in a new folder of the computer so Matlab can access it later on.

This processing is not only done to the training and testing images, but also for any dataset from which the crop and weeds are to be identified. The only difference between both situations is the name of the folder where the images are to be saved after the processing.

```
5 - path = 'C:\Users\linac\Desktop\TFG\Final_set';  
6 - ds = imageDatastore(path, 'IncludeSubFolders', true);  
7 - num = numel(ds.Files);
```

**Figure 79:** Loading images to a datastore.

```

11 - for i = 1:1:num
12 -
13 -     im = readimage(ds,i);
14 -     imgray = rgb2gray(im);
15 -     bin = imgray > 225;
16 -     binary = imdilate(bin, true(1));
17 -     red = im(:,:,1);
18 -     green = im(:,:,2);
19 -     blue = im(:,:,3);
20 -
21 -     red = regionfill(red, binary);
22 -     green = regionfill(green, binary);
23 -     blue = regionfill(blue, binary);
24 -
25 -     rgbimage = cat(3, red, green, blue);
26 -     im = imsharpen(rgbimage, 'Radius', 2, 'Amount', 1.65);
27 -
28 -     filename = sprintf('%s_%d.%s', 'C:\Users\linac\Desktop\TFG\Processed_Images\processed_', i, 'jpg');
29 -     imwrite(im, filename);
30 -
31 - end

```

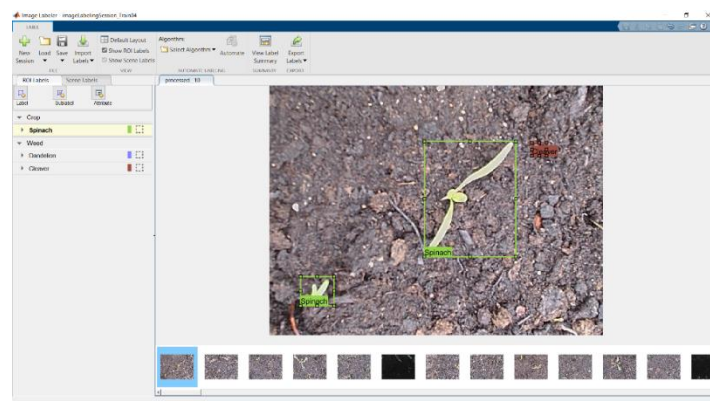
**Figure 80:** Image processing

### 3.2 Network training

When all the images from training and testing datasets are processed, it is time to train the network. The full code is shown in *Figures 87 and 88*. In this section of the appendix, an explanation of each of the steps taken to train and test the network is done in detail.

The first step is to take the processed images and create a new datastore. This datastore is randomly split in two sets of images: 'imTrain' and 'imTest'. In this thesis, the 70% of the images are part of imTrain and the rest of them will conform imTest.

To train the network into object detection, it is necessary that the net knows which are those objects and where to find them in the pictures. That is why the next step in the training process is to label the objects inside the different set of images. To do so, the ImageLabeler application (*Figure 81*) is used in Matlab. By using this app, a table called 'ground truth', where all the crops and weeds are in each photo, is saved; later, this table will be used for training, testing and even proving if the predictions are correct or not.



**Figure 81:** ImageLabeler app



Once the datasets are labelled and their ground truth table is done, the training of the network can start. The train ground truth table is given format in order to make it processable by the net as shown in *Figure 82*.

```
47 %% Training set groundtruth
48
49 - traindata = objectDetectorTrainingData(gTruth_Train);
```

**Figure 82:** Formatting ground truth table.

The net is loaded into the program and two out of three of its last layers are modified as it can be seen in *Figure 83*. This is done because the network is performing Transfer Learning. Layer 23 is modified so it can recognize four different objects: spinaches, cleavers, dandelions and the background. Layer 25 is also modified, but this one is left blank for it to classify.

```
51 %% Network modification
52
53 - net = alexnet;
54 - layers = net.Layers;
55 - layers(23) = fullyConnectedLayer(4);
56 - layers(25) = classificationLayer();
```

**Figure 83:** Network modification

Before finally training, some training options have to be stated. In the case of the RCNN being trained in this thesis, the options are the training method the mini batch size, the maximum number of epochs and the initial learn rate. To decide these options, a study has been made on them:

- Method: there are two main training methods: 'adam' (Adaptive Moment Estimation) and 'sgdm' (Stochastic Gradient Descent). 'Sgdm' computes the training on a small random subset of the data introduced in each repetition, performing as gradient descent with low learning rate. 'Adam' adapts its learning rate for each weight of the neural network depending on how they evaluate some small random data. In order to choose one of them, a training was done with each one. Finally, 'Adam' performance was much better than 'sgdm', so it was chosen as the training method.
- Mini batch size: sets the batch sizes to be introduced to the net while training. By doing some research, it was found that the optimum size to start with Adam method was 64, and after testing it, 64 was stated as the final mini batch size.
- Maximum number of epochs: sets the number times the whole training set is used. To estimate the optimum number some trainings were done. The final value was set choosing the highest that avoids overfitting, trying not to have incrementing loss. This value was finally 20.
- Initial learn rate: sets how quickly the weights are updated during training, adapting the net to new data. The lower it is, the slower the net will learn, but the higher it is, the easier it is to

converge to suboptimal solutions.  $3 \cdot 10^{-4}$  is set as the initial learn rate after some research done in similar projects using Adam method.

With the net loaded and modified, the ground truth data of the images extracted, and the training options settled, the network training is performed, as seen in *Figure 84*.

```
58 %% Training options
59
60 - options = trainingOptions('adam', 'MiniBatchSize', 64, 'MaxEpochs', 20, 'InitialLearnRate', 3e-4);
61
62 %% RCNN Training
63
64 - [rcnn,info] = trainRCNNObjectDetector(traindata, layers, options);
65
```

**Figure 84:** Training options and network training

Once the network is trained, the training progress can be plot in case it is needed. The code to plot it is shown in *Figure 85*, where two graphs are shown, one representing the accuracy depending on what iteration it is and the other one the loss.

```
66 %% Training progress
67
68 - figure
69
70 - subplot(2, 1, 1)
71 - title('Training Progress')
72 - plot(info.TrainingAccuracy(1, :), 'b')
73 - ylabel('Accuracy (%)');
74 - xlabel('Iteration');
75
76 - subplot(2, 1, 2)
77 - plot(info.TrainingLoss(1, :), 'r')
78 - ylabel('Loss');
79 - xlabel('Iteration');
```

**Figure 85:** Commands to plot the training progress

Then comes the testing. After the network has been properly trained and the results are good enough, a test step is done in order to observe how the network performs with a set of images that has not been introduced to it before. The results the network gives to the imTest set of images, are stored in a table. Those results are the coordinates of the bounding boxes where the detected objects are, and a score, which is how precise that prediction is.

Then, if the prediction is higher than a threshold value given, in this case it is the 70% of accuracy, when the images are shown, the bounding boxes with the proper labels are also shown in those pictures. All the code related to this part, is shown in *Figure 88*.

After having tested the whole set of images, an evaluation on how the network has performed is done as shown in the *Figure 86* below.

```
107 %% Evaluation
108
109 - overlap = 0;
110 - apS = evaluateDetectionPrecision(results, gTruth_Test.LabelData(:,1), overlap);
111 - apD = evaluateDetectionPrecision(results, gTruth_Test.LabelData(:,2), overlap);
112 - apC = evaluateDetectionPrecision(results, gTruth_Test.LabelData(:,3), overlap);
113
```

**Figure 86:** Evaluation of the test set of images code

```
38 %% Splitting Datastores
39
40 - imds = imageDatastore('C:\Users\linac\Desktop\TFG\Processed_Images','LabelsSource', 'folderNames');
41 - [imTrain, imTest] = splitEachLabel(imds, 0.7, 'randomized');
42
43 %% Open Image Labeller App.
44
45 - imageLabeler
46
47 %% Training set groundtruth
48
49 - traindata = objectDetectorTrainingData(gTruth_Train);
50
51 %% Network modification
52
53 - net = alexnet;
54 - layers = net.Layers;
55 - layers(23) = fullyConnectedLayer(4);
56 - layers(25) = classificationLayer();
57
58 %% Training options
59
60 - options = trainingOptions('adam', 'MiniBatchSize', 64, 'MaxEpochs', 20, 'InitialLearnRate', 3e-4);
61
62 %% RCNN Training
63
64 - [rcnn,info] = trainRCNNObjectDetector(traindata, layers, options);
65
```

**Figure 87:** Network training (I)

```

81 %% Test
82
83 results = table('Size', [numel(imTest.Files) 2], 'VariableTypes', {'cell', 'cell'}, 'VariableNames', {'Boxes', 'Scores'});
84
85 for j = 1:numel(imTest.Files)
86
87     [dbox, dscore, dlabel] = detect(rcnn, imread(imTest.Files{j}));
88
89     results.Boxes{j} = dbox;
90     results.Scores{j} = dscore;
91
92     idx = dscore > 0.7;
93     dboxTop = dbox(idx, :);
94     dlabelTop = dlabel(idx);
95
96     if isempty(dlabelTop) == 0
97         detected = insertObjectAnnotation(imread(imTest.Files{j}), 'rectangle', dboxTop, cellstr(dlabelTop));
98         figure
99         imshow(detected)
100     else
101         figure
102         imshow(imTest.Files{j})
103     end
104 end
105
106
107 %% Evaluation
108
109 overlap = 0;
110 apS = evaluateDetectionPrecision(results, gTruth_Test.LabelData(:,1), overlap);
111 apD = evaluateDetectionPrecision(results, gTruth_Test.LabelData(:,2), overlap);
112 apC = evaluateDetectionPrecision(results, gTruth_Test.LabelData(:,3), overlap);
113

```

**Figure 88:** Network training (II)

### 3.3 Plant detection

Figure 89 shows an overview of the code that corresponds to the plant detection, parting from a new set of images unknown to the network. The pictures are loaded in Matlab and processed the same as explained in point “Image processing”. This is represented from line 96 to 122 of the figure below, where the compressed ‘for’ function is the processing. Next, a new datastore is created to load the processed pictures into Matlab, and three variables are created: ‘results\_pred’, ‘h’ and ‘k’. Lines from 130 to 161 are compressed in a ‘for’ function, where the plant detection and coordinate obtention will be done. Finally, the evaluation of the detection is done, from which the precision of the network for each category is obtained. Below, an explanation of the most important parts of the code will be done.

```

96 - path_pred = 'C:\Users\linac\Desktop\TFG\Prediction';
97 - ds_pred = imageDatastore(path_pred, 'IncludeSubFolders', true);
98 - num_pred = numel(ds_pred.Files);
99
100 % Processing
101
102 - for i = 1:1:num_pred...
103
104     imds_pred = imageDatastore('C:\Users\linac\Desktop\TFG\Processed_Prediction');
105
106 % Detection
107 results_pred = table('Size', [numel(imds_pred.Files) 2], 'VariableTypes', {'cell', 'cell'}, 'VariableNames', {'Boxes', 'Scores'});
108 h = 1;k = 1;
109 clear coord;
110
111 - for j = 1:numel(imds_pred.Files)...
112
113 % Evaluation
114
115 overlap = 0;
116
117 apS_pred = evaluateDetectionPrecision(results_pred, gTruth_TestMod.LabelData(:,1), overlap);
118 apD_pred = evaluateDetectionPrecision(results_pred, gTruth_TestMod.LabelData(:,2), overlap);
119 apC_pred = evaluateDetectionPrecision(results_pred, gTruth_TestMod.LabelData(:,3), overlap);
120

```

**Figure 89:** Plant detection code overview

As it has been previously mentioned, image processing can be found in *Figure 80* since it is the same code for every image set used in this project.

Starting then with *Figures 90* and *91*, they show the code that corresponds to the object detection in each image and the coordinate localization for each weed. For each element of 'imds\_pred' (meaning for each image of the datastore), the code runs the 'detect' command, with the trained network named 'rcnn'. This command returns the bounding boxes ('dbox') for each plant detected in the image, as well as the confidence ('dscore') and the label ('dlabel') assigned to each bounding box. These results are stored in the table 'results\_pred' created earlier, which will be of use later on during the evaluation. Next, a thresholding is done to the score value, and only the detected elements with a confidence value greater than 75% are considered for the next part.

If the image has no label assigned to any object, the program will print the original image with no annotations. If, on the other hand, there is at least one valid label for the image, the code calls the function 'Coordinates', stores the coordinates of the objects that are not labelled as spinach in 'coord' and shows the image with the corresponding bounding boxes and labels.

```

131 - for j = 1:numel(imds_pred.Files)
132 -
133 -     [dbox, dscore, dlabel] = detect(rcnn, imread(imds_pred.Files{j}));
134 -     results_pred.Boxes{j} = dbox;
135 -     results_pred.Scores{j} = dscore;
136 -     idx = dscore > 0.75;
137 -     dboxTop = dbox(idx, :);
138 -     dlabelTop = dlabel(idx);
139 -
140 -     if isempty(dlabelTop) == 0
141 -         for i = 1:(numel(dboxTop)/4)
142 -             if dlabelTop(i) ~= 'Spinach'
143 -                 [coord_x, coord_y] = Coordinates(dboxTop, i, k, c);
144 -                 coord{h,2} = [coord_x, coord_y];
145 -                 coord{h,1} = dlabelTop(i);
146 -                 h = h+1;
147 -             end
148 -         end
149 -         detected = insertObjectAnnotation(imread(imds_pred.Files{j}), 'rectangle', dboxTop, cellstr(dlabelTop));
150 -         figure
151 -         imshow(detected)
152 -     else
153 -         figure
154 -         imshow(imds_pred.Files{j})
155 -     end
156 -
157 -     c = c + 1;
158 -
159 -     if mod(j,6) == 0
160 -         k = k+1;
161 -         c = 1;
162 -     end
163 - end

```

**Figure 90:** Code for object detection and coordinate obtention.

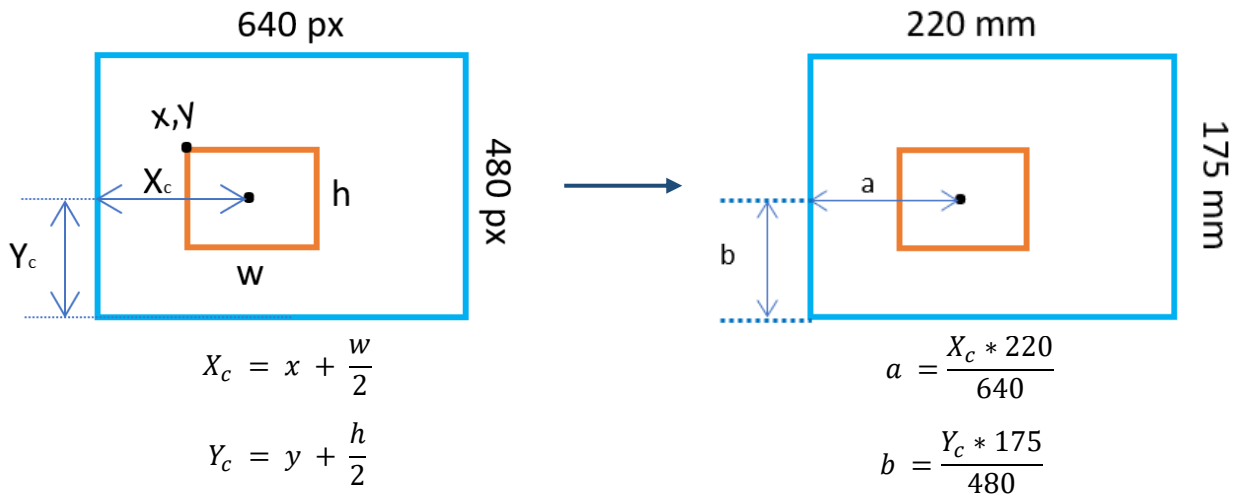
```

1 - function [coord_x, coord_y] = Coordinates (dboxTop,i, k, c)
2 -
3 -     xc = (dboxTop(i, 3)/2)+dboxTop(i, 1);
4 -     yc = (dboxTop(i, 4)/2)+dboxTop(i, 2);
5 -
6 -     a = xc*220/640;
7 -     b = yc*175/480;
8 -
9 -     coord_y = 1320-(220-a)-(220*(c-1));
10 -    coord_x = 1575 - (175-b)-(175*(k-1));

```

**Figure 91:** Function Coordinates

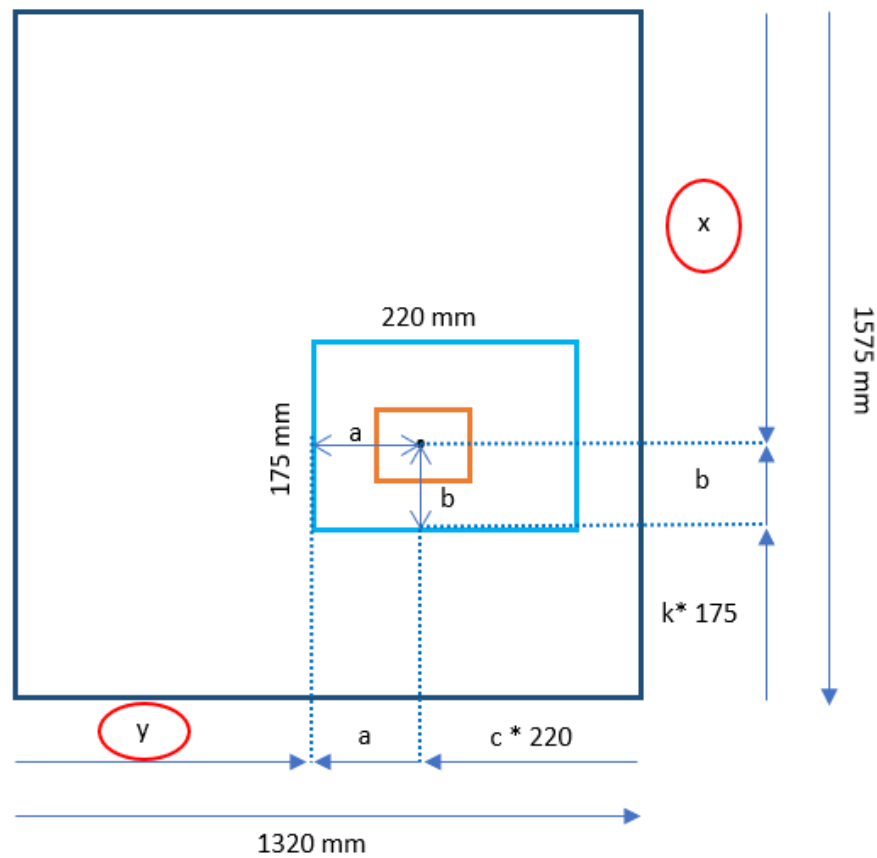
The function 'Coordinates' used in line 143 of *Figure 90* is shown in *Figure 91* and explained in more detail in the figures below. The first step is calculating the coordinates of the center of the bounding box, as explained in *Figure 92*. After some transformations to new coordinate systems, this point will represent the location of the weed.



**Figure 92:** Obtention of bounding box center

From line 3 to 10 of *Figure 91*, the coordinates of the center of every bounding box according to FarmBot's coordinates system are calculated. As the images obtained from FarmBot are sorted from latest to oldest, the first picture corresponds to the one located at the lower-right corner of the FarmBot bed. As there are 6 pictures per column and 9 per row, the X coordinate of the image will only vary every sixth picture (represented by variable 'k' in *Figure 91*) while the X coordinate varies with every column (represented by variable 'c' in *Figure 91*).

As the image is 640 x 480 pixels wide and high, and the area captured by the picture is 220 x 175 millimeters wide and high respectively, to obtain the correct coordinates of the weed, the location of the bounding box center, is transform as shown in *Figure 92*. Once the center is obtained, its location and the picture location in terms of x and y axis are subtracted from the FarmBot dimensions, getting the final weeds coordinates according to the FarmBot coordinate system. A visual representation of the previous explanation as well as the FarmBot bed dimensions are shown in *Figure 93*.



**Figure 93:** Final coordinates according to FarmBot

The network evaluation in this part of the project would not be necessary for the use of this weed detector system, but as in this thesis it is important to show the performance of the new system, it has been added as a final step. *Figure 94* shows the commands used previously modifying the inputs and outputs to correspond to the prediction step instead of the testing done in previous chapters.

```
% Evaluation
overlap = 0;
apS_pred = evaluateDetectionPrecision(results_pred, gTruth_prediction.LabelData(:,1), overlap);
apD_pred = evaluateDetectionPrecision(results_pred, gTruth_prediction.LabelData(:,2), overlap);
apC_pred = evaluateDetectionPrecision(results_pred, gTruth_prediction.LabelData(:,3), overlap);
```

**Figure 94:** Prediction evaluation