Precision Farming

January 19, 2024

Name: Izuchukwu George Enekwa
Email: izuchukwu-george.enekwa@stud.hshl.de

Name: Nnaemeka Valentine Eze
Email: nnaemeka-valentine.eze@stud.hshl.de

Name: Jires Donfack Voufo
Email: jires.donfack-voufo@stud.hhl.de

Name: Abdullah Al Forkan
Email:Abdullah-al.forkan@stud.hshl.de

Abstract

Precision farming is a new concept that uses cutting-edge information technology to boost agricultural output and effectiveness. By utilising the latest advancements in automation, artificial intelligence, and networking, farmers can monitor every stage of the process more effectively and apply precise treatments that are precisely determined by machines with remarkable precision. Farmers, data scientists, and engineers are still working on ways to maximise the amount of manpower required in agriculture. When essential information resources improve, smart farming transforms into a learning system that learns every day and gets smarter. We have used two smart farming approaches in this study and its practical application: a timed Petri Net, which enables real-time modelling of our use cases in resolving agricultural issues, and a deep learning approach, which is a machine learning method based on the principles of an artificial neural network. Our primary model for training and analysing our datasets was the CNN model, which provided us with an accuracy of 98 percent over 10 epochs. Weed identification, plant health monitoring, soil moisture measurement and watering, and weed detection were all enhanced by our method. Research can still be done in the future to preserve and enhance our strategy.

Contents

List of Figures

1 Introduction

Precision farming involves monitoring, measuring, and adapting to changes in both space and time based on need.[3] It is a sustainability farming management approach and could be employed in the production of both crops and animals. Precision farming frequently uses technology to automate farming processes, enhancing their diagnostic, decision-making, or execution of it. The goal of precision farming practice is to enhance the decision-making system for farm management in other to increase yield, long-term viability, and sustainability.

This is in contrast to traditional agriculture which is characterized by a predetermined schedule for performing specific tasks, such as planting or harvesting; irrespective of the prevailing circumstances. Traditional farmers for instance instinctively apply fertilizers or pesticides without precaution, even when there may not necessarily be the need for such. These could lead to environmental pollution. Also in planting, some farmers tend to spray their seedlings with little or no consideration for the spatial requirements of different plant species. The resultant effect is the growing and harvesting of plants that are either malnourished or disease-infested. There are equally many instances where traditional agricultural practices could be improved for the good of man and his environment. Hence these needs necessitated a more systematic and predictive model in Agriculture. This Predictive analysis/approach could be called Precision farming. In the 1980s, the idea of precision farming was originally developed in the United States.[4]. It involves forecasting, planning, and managing agricultural practices through the use of satellite-based technology and information technology (IT). These may include the use of robotic systems like Unmanned Aerial Vehicles(UAV), and various AI tools to achieve the tasks above.

In this project, we have designed and implemented a precision farming prototype using deep learning algorithms in conjunction with the Cyber-Physical System (CPS) idea to effectively carry out various farming operations. In order to grasp the need, the trends, and to make well-informed decisions regarding them, we suggested engaging in activities that entailed first analyzing situations and farm needs, then obtaining relevant data. We concluded by analyzing various data using a Deep learning model and making predictions capable of monitoring the health of plants.

2 System analysis

Our overall aim is to develop a simple prototype that could be used to model an autonomous system, capable of carrying out multiple tasks in a farm setting, using precision farming. At first, with a simple framework for the various scenarios, and improved upon to accommodate other use cases and system features. The use case is made of main actors; the farmer, secondary actors, and the main system. The Farmer would power the Surveillance drone and equally oversee the entire system. The Surveillance drone is assigned the chief responsibility of

mapping out the entire system by navigating through the farm, and making a Survey of the overall farm space. These information are made to be captured in the information system unit/database.

The system can be classified into three layers;

- The Input/sensing Layer
- The Network Layer
- The Output/Service Layer

The picture bellow depict the achitecture of the system

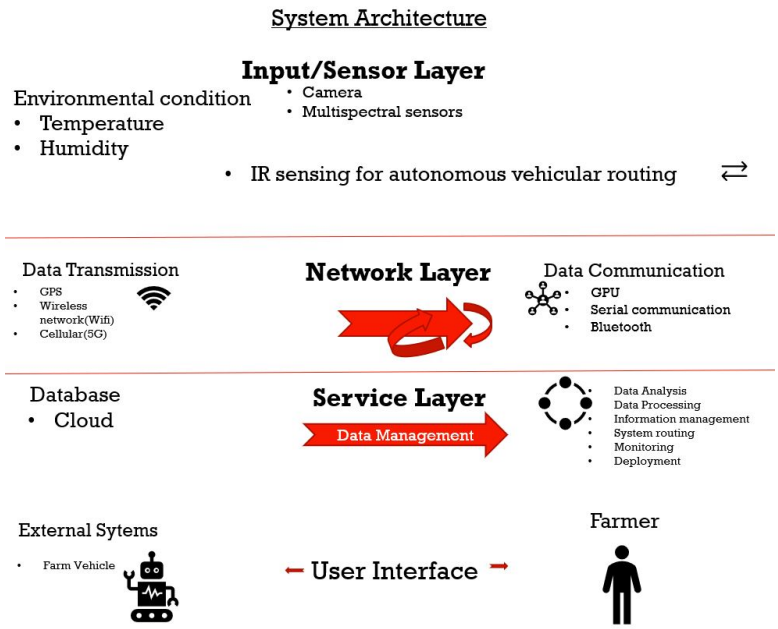

Figure 1: System Architecture

- **Data Collection and Management**
  In this project, the surveillance drone is charged with the responsibility of mapping out the entire farm space. With a GPS mounted on the surveillance drone, it could be able to carry out the above-mentioned task. It is equally responsible for getting data from crops as well as the environment. The surveillance drone uses imaging devices and sensors to capture the data that are sent to the communication. For instance, it captures the image of plant leaves that would be used in deep learning for model training. The drone equally scans for weeds, checks for water level as well and detects a plant that needs treatment. These tasks could be handled by different data collection drones, as we tried earlier. However, in other

to reduce the cost of maintenance, we finally settled for one drone for this task since it can do that effectively.

- **Communication Infrastructure and Networks Connectivity**
  In precision farming, the role of communication infrastructure in data collection and processing can not be over-emphasized. These may include the sensor network, as well as wireless communication. For the service and surveillance drone to be able to work together in real-time, there is a heightened need for synchronous communication . This is why we used a timed petri-net to enhance the communication. The system gets data from the wireless network as well as the cloud for information management. Part of the decision-making we faced was how the data was collected for analysis. Our system is equipped with a central database, where all captured data are registered. This central database was connected to a cloud. The deep learning model can be trained with these data simply by specifying the data path or downloading it onto a storage device.

.

3 Use Case

3.1 Field Mapping

- **Check Size of the farm** The drone collects information related to the topography and the boundaries of the field.

- **Soil Monitoring**
  By Measuring the Soil moisture and Humidity, we give the farmer the ability to use water in a more efficient way. This is even more important in regions where the climate is hot and water resources are limited.

- **Scan for Weed and Sick Plant**
  Weed infestation represents a threat because for the crop, it can disrupt the growing process in this case of the sugar beet.Diseases can affect significantly the production output in agriculture.That is why having a drone that can quickly fly over the field to identify which plant are sick is crucial.This give the farmer the ability to react in an efficient way to avoid any food crisis.

- **Save Data**
  After collecting all information related to the dimension of the field, state of the soil, and weed infestation the first drone sends those to the server.

.

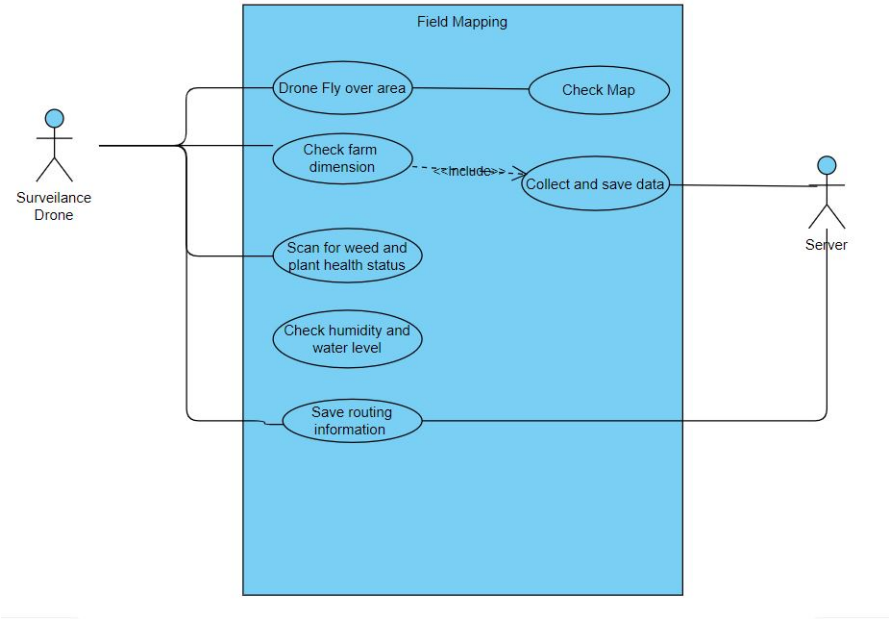

Figure 2: Field Mapping

3.2 Weed Destruction

The drone sprays herbicide to eliminate any plant that is considered as weed.

- **Data collection**
  In this first step, the 2nd drone collects from the server data related to which plant was detected as weed and also where exactly it is located.
- **Area Location**
  In this second step, Using the coordinate of the location collected from the server, the 2nd drone directly flies to a specific area and does not have to scan the whole field again.
- **Weed and Sugar beet detection**
  In this third step, The drone gets to the spot where the weed is supposed to be and double checks if the first drone detected correctly which planted is weed. .

.

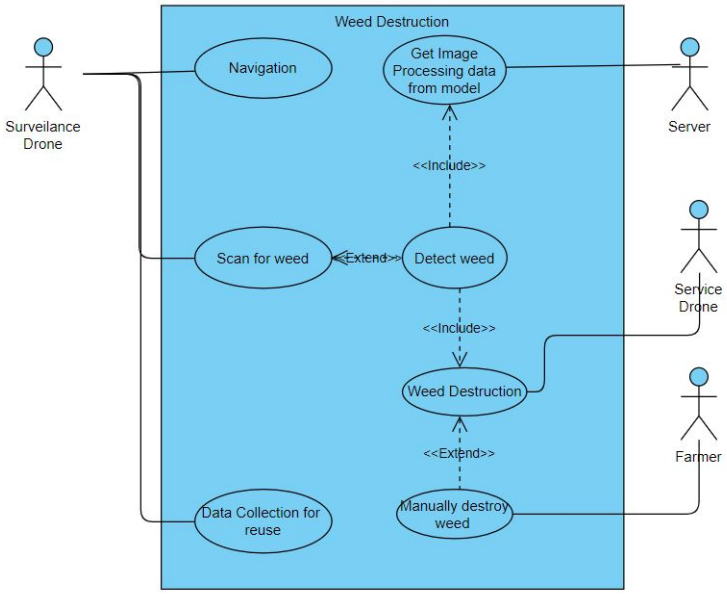

Figure 3: Weed Detection

3.3 Sick Plant treatment

- **Data collection**
  In this first step, the 2nd drone collects from the server data related to which plant waste deemed sick and also where exactly it is located.

- **Area Location**
  In this second step, Using the coordinate of the location collected from the server, the 2nd drone directly fly to a specific area and does not have to scan the whole field again.

- **Identify Sick plant**
  In this third step, The drone gets to the spot where the sick plants are and double checks if the first drone detected correctly which plant is disease-ridden.

- **Treatment of the plant**
  In this fourth step, the drone sprays some chemicals that will stop the spreading of the disease. By conducting the treatment in a restricted area we also reduce the amount of herbicide needed.

.

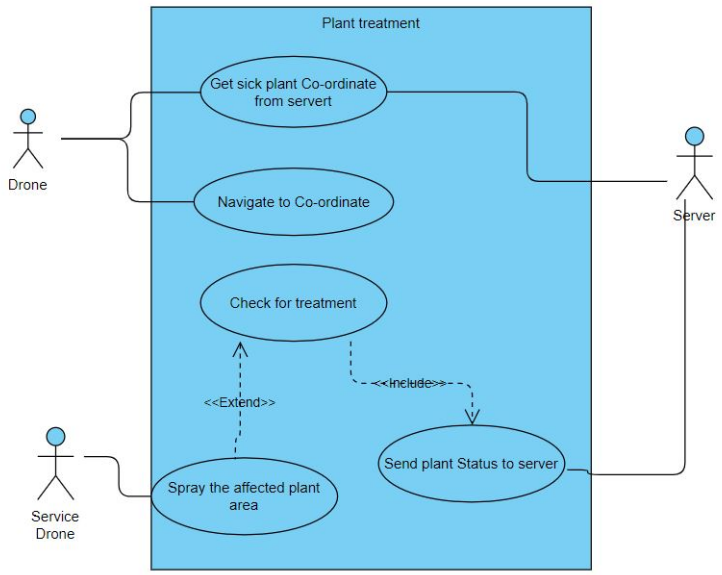

Figure 4: Sick plant Treatment

3.4 Plant watering

- **Data collection**
  In this first step, the 2nd drone collects from the server data related to where the soil is too dry and the plant is at risk of dying.

- **Locate the area**
  In this second step, Using the coordinate of the location collected from the server, the 2nd drone directly fly to a specific area and does not have to scan the whole field again.

- **Spray the water**
  In this third step, the drone will spray some water on the dry soil based on the needs of the plant. This will provide the crops with the needed amount of water they need to grow.

.

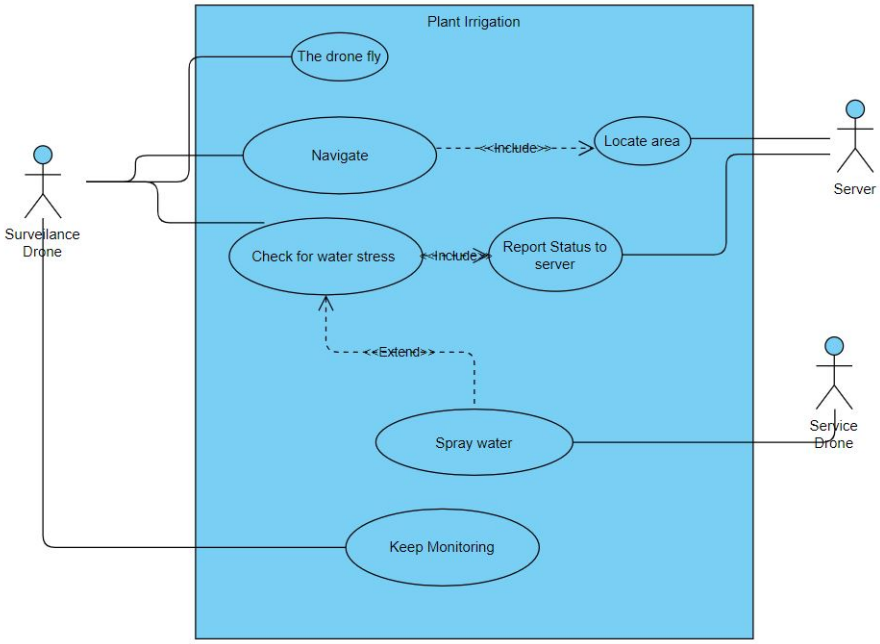

Figure 5: Plant Watering

4 System Model

Once our use case was established, we used TAPAAL to model and validate Timed-Arc Petri Nets (TAPN) in order to validate the proper flow of the process of our system.[1] It consists of a graphical editor, simulator, verification environment, query interface as well as query dialog.

Our Tapaal model was directly mapped out of our use case diagram. As explained in the section above. Our system includes two drones, one named the surveillance drone, and the second as the duty drone.

4.1 Tapaal model for surveillance

The Tapaal model of the surveillance drone as seen in figure 6 Shows the three scenarios or tasks which our drone intends to carry out. The weed destruction, irrigation system, and plant health monitoring system. The surveillance drone will scan the map area looking for weeds, low soil moisture, and sick plants. The transitions T0, T4, and T5 represent these scenarios, which will fire with the time interval 0 to 1. When one of these is detected or found, the information is then sent to the save location. To save the locations, the transitions T1, T6, and T7 must fire. When the location has been saved, it is then transported to the server, where the information is saved for the duty drones to pick.

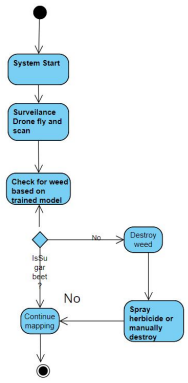

Figure 6: Activity Diagram-Weed destruction

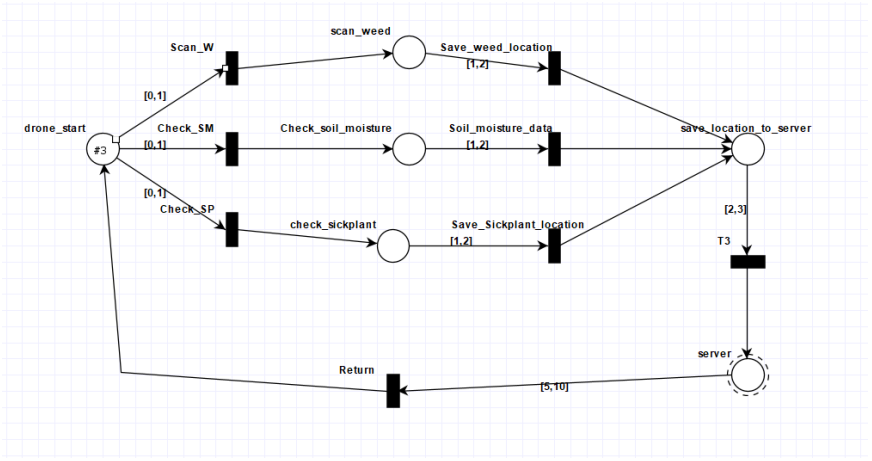

Figure 7: Surveillance Drone

4.2 Shared server (Tapaal place)

The system has a shared server where all the information captured by the surveillance drone is saved. An explanation of how this works is explained in the sections below. There are also time constraints with the time intervals from assigned to them.

4.3 Tapaal Weed destruction

The surveillance drone and duty drone use shared data for carrying out tasks. For example, when the surveillance drone system detects weed, the coordinates and location of the weed on the farm area will be saved to the server, where the duty drone in charge of weed destruction can pick up this information and

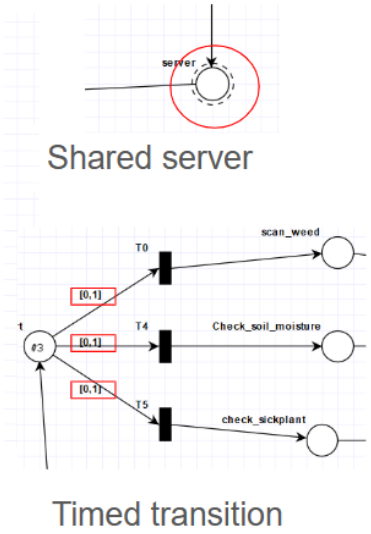

Figure 8: shared server and time delay

proceed to destroy the weed.

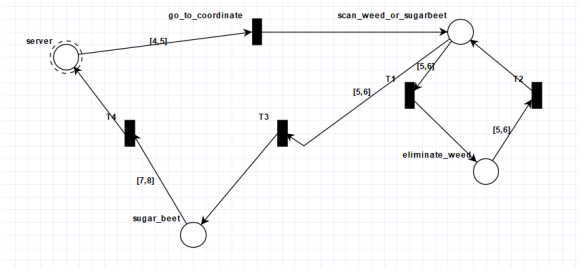

Figure 9: Weed destruction

4.4 Tapaal model for sick plant detection

After a sick plant is detected, the location of the sick plant is saved to the server. The Tapaal model from fig 9 is a system model for the sick plant treatment drone.(duty drone) from the Tapaal model,the system starts with pickup sick plant coordinate, then proceeds to locate coordinate, after the drone has arrived at the location, it checks for lesion on the plant leaf, if lesion is detected, the transition treat plant fires, but if no lesion is detected “false” the system returns the information to the server.

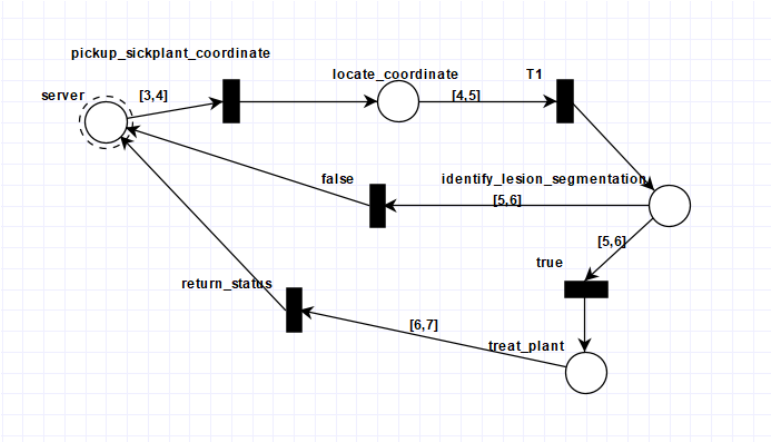

Figure 10: Sick Plant treatment

4.5 Tapaal model for irrigation system

The model is supposed to check if the soil moisture or humidity is low. It also checks if there is rain in the forecast, If there is no rain in the forecast of humidity and soil moisture are low then the duty drone proceeds to water the farm area. The duty drone for the irrigation system just like other duty drones picks up this information from the server. Then proceeds to water the farm. The transition T2 gets coordinated, T3 navigates, and T4 sprays water. T5 returns the status back to the server.

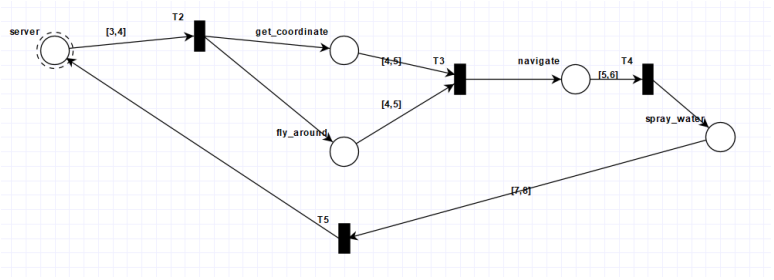

Figure 11: Irrigation

5 Verification

Query Interface- The query interface uses the CTL as described above to verify that a certain condition is met in the system behaviour. It is noteworthy that the overall aim of the verification is not to have a deadlock; but to avoid it. Let's take one of the properties and illustrate. AF(For All Finally): This logic

indicates that a given property holds for every path that a system may take and that the property will ultimately come to pass.

5.1 Reachability

Reachability in Petri nets is the capacity to go from an initial state to a certain state. In a Petri net model, reachability analysis is performed to ascertain if a particular state can be reached.

Places and transitions are the two primary parts of a Petri net. Transitions describe events or possible activities that can take place in the system, whereas places represent states or circumstances. Tokens are used to show if something is present or absent at a location. They are frequently represented by dots.

Tokens are distributed across the locations at the beginning of a reachability Petri net, known as the first marking. Through the application of the firing rules, which define the possible locations and transitions for tokens, we were able to model the system's behaviour and explore various states.

We were able to ascertain whether a particular marking or state is attainable from the initial marking through reachability analysis. This is accomplished by determining if a firing sequence of transitions can result in the intended state. The state is attainable if such a sequence occurs; else, it is not.

These are a few of the questions that were used to test the reachability of the model.
If markers are accessible within the network
If every marking satisfies the specified attribute
Does the reached marking satisfy after firing?

5.2 Deadlock check

We examined the reachability of deadlock states in order to carry out a deadlock check in our system. When every transition is disabled, there aren't enough tokens in the locations for any transition to be able to trigger, which results in a deadlock situation.

There are various ways to examine a Petri net for stalemate. A popular method is to leverage the liveness attribute known as "boundedness." If there is an upper limit on the total number of tokens at each location, the Petri net is said to be bounded. In the event that a Petri net is unbounded, a deadlock situation could occur.

We were able to learn more about the behaviour of the system and spot possible problems that can occur when it is being executed by carrying out a deadlock check.

5.3 Trace checking

Trace checking is a method used in Petri nets to confirm a system's behaviour by looking at the possible transition sequences.

In Petri net terminology, a trace is a series of transitions that have the ability to be enabled and fired in a particular order. Analysing these sequences to make sure they adhere to specific restrictions or attributes is known as trace checking.

A frequent attribute that can be examined by trace checking is a state's reachability. We can ascertain whether it is feasible to go from the starting state to the desired state by looking at the traces in a Petri net.

Another further attribute that can be verified is the lack of specific undesirable conditions. We can make sure the Petri net doesn't enter states that go against particular requirements or limitations by looking at the traces.

We can utilise algorithms that investigate potential transition sequences in the Petri net to do trace checking. These algorithms can assist us in determining whether a particular undesirable state can be attained or if a desired state can be achieved.

Here, we made sure to check If the markings on the input and output are the same (to check if a certain transition sequence can be enabled and carried out),

6 Deep Learning Image Processing

Deep learning for image identification is the process of automatically learning and extracting features from photos using multiple-layered neural networks, or deep neural networks. By enabling computers to perceive and comprehend visual data, this technique has greatly advanced the science of computer vision. This when utilized, is the focal point in precision farming, as illustrated in this project using Fastai framework on a Jupyter Notebook environment.

The project entails several data processes, model training and interpretation, and testing, as well as segmentation. However, the emphasis here is on the deep learning algorithm used, as well as the analysis that yields the result provided below.

6.1 Tools Used

We used Fastai library to make the image recognition system with Python in jupyter notebook, which makes image recognition very easy. It required enough GPU power to handle the large image dataset.

6.2 Library Setup

The fastai vision library provides all the tools necessary for the implementation. It can called with a simple code line,

```
from fastai.vision.all import *
```

This imports all the necessary modules from fastai library for computer vision tasks. The * symbol is a wildcard character that imports all the modules in the fast.vision package.

6.3 Geting Training Dataset

In deep learning, a dataset entails a collection of examples that are used to train, validate, or test the model. The dataset serves as the foundation for training a model, with a view to learning features as well as making predictions. This is because, it serves as an input to a deep learning model, from which patterns and features are learned.

Our dataset contains about 4516 labeled images from which our model would be learning to identify.

```
'Black-grass', 'Charlock', 'Cleavers',
'Common Chickweed', 'Common wheat',
'Fat Hen',
'Loose Silky-bent', 'Maize',
'Scentless Mayweed',
'Shepherd's Purse', 'Small-flowered
Cranesbill',
'Sugar beet'
```

The whole dataset call be loaded into the system with the following code,

```
path = '/folder/path'
fnames = get_image_files(path)
fnames
```

Here, the path variable collects the training image dataset from the folder. which is the directory from which you want to get the image files. The fnames variable is then set to the list of image files returned by the 'get image files' function.

6.4 Data Block and Data Loader

Data block is one of the most important part of any image recognition system. This sets the configuration for the images. We created the following configuration in the data block,

```
plants = DataBlock(
    blocks = (ImageBlock, CategoryBlock),
    get_items = get_image_files,
    get_y = parent_label,
    splitter = RandomSplitter(seed = 42),
    item_tfms = Resize(460),
    batch_tfms = aug_transforms(size=224, min_scale=0.75)
)
```

The code creates a DataBlock object that can be used to load images of plants from a directory. The blocks parameter specifies the types of data that the DataBlock object will work with. In this case, the DataBlock object will work with imageBlock, and for single-label classification we use CategoryBlock. The 'get items' parameter specifies a function that returns a list of image files from

the directory specified by the path parameter. The 'get y' parameter specifies a function that returns the label as folder name for a given image file. The splitter parameter specifies a function that splits the data into training and validation sets, in the commonly accepted ratio 80:20. The 'item tfms' parameter specifies a set of image transformations that are applied to each image. The 'batch tfms' parameter specifies a set of transformations that are applied to each batch of images. Then we assigned the data block with the data loader using the following code,

```
data_loader = plants.dataloaders(path)
```

Here, the plants variable is a DataBlock object that specifies how the images should be loaded and transformed. Then the dataloaders method is called on the DataBlock object to load the images into memory in batches.

6.5 Building A Model

Our main aim is to train a model capable of identifying different plants and segmenting them to be able to be used in the future for other data-related decision-making. As described in our use cases, these data were meant to check for weeds in the farm using binary classification for Sugar beet. Considering the scenario described, we used pre-trained ResNet50 architecture for this task. This is because it has been proven to be very efficient in image classification, especially when dealing with large image datasets.[2] ResNet50 is a variant of CNN(Convolutional Neural Network) which is known for its use of residual connections by bypassing some layers to achieve better results. This feature, as well as other preprocessing techniques like augmentation, helps it to minimize overfitting.[2] The model is made to learn the pattern using the learner function as described below in the implementation code.

```
learner = vision_learner(data_loader, resnet50, metrics=error_rate)
learner.model
```

Inside the learner 'data loader'variable specifies how the images should be loaded and transformed. The 'error rate' function is used as a metric to evaluate the performance of the model.

6.6 Learning

```
learner.fit_one_cycle(10)

print("Accuracy: ", 1 - float(learner.recorder.metrics[0].value))
learner.lr_find()
```

This ensures that the learnings were made better by repeating the task for 10 epochs. After the model had successfully learned the image patterns, we really

needed to know the extent to which it had learned thereby printing the accuracy. This returned about 96 percent accuracy, which shows that the system was able to learn the pattern satisfactorily, and is ready for deployment into other tasks. The 'fit one cycle' method is used to train the Resnet50 model using the fastai library. The learner object specifies how the model should be trained and the 10 parameter specifies the number of epochs for which the model should be trained. Some important terms here to note are that each epoch means one forward and backward pass of all training samples and Batch size is the number of training samples in one epoch. This prints the accuracy of the model on the validation set. The '1-' part is used to convert the validation loss to accuracy. Finally, 'lr find' is used to find a good learning rate for training.

```
learner.fit_one_cycle(10)
```

epoch	train_loss	valid_loss	error_rate	time
0	1.576195	0.430090	0.142234	01:16
1	0.801373	0.345520	0.103839	01:14
2	0.485515	0.278417	0.089005	01:16
3	0.370402	0.208551	0.075916	01:15
4	0.298338	0.177664	0.053229	01:13
5	0.257325	0.138703	0.048866	01:12
6	0.209454	0.127272	0.044503	01:12
7	0.177145	0.121725	0.041885	01:11
8	0.154366	0.122826	0.040140	01:11
9	0.141897	0.120050	0.036649	01:13

```
print("Accuracy: ", 1-float(learner.recorder.metrics[0].value))
```

```
Accuracy:  0.9633507840335369
```

Figure 12: Learning

6.7 Interpretation and Export

Interpretation is very usefull to check the inconsistency of the images. This shows us the top losses happened while learning. We used the following code for interpretation,

```
interpretation = ClassificationInterpretation.from_learner(learner)
losses,indexes = interpretation.top_losses()
len(data_loader.valid_ds) == len(losses) == len(indexes)
interpretation.plot_top_losses(8, figsize=(22,13))
```

This creates an interpretation object that can be used to interpret the predictions of a neural network model trained using the fastai library. The 'top losses' method of the Interpretation object is then called to get the top losses and their corresponding indexes. The 'len' statement checks if the number of items in the validation set is equal to the number of losses and indexes. Then we plotted the Interpretation using the last code line.

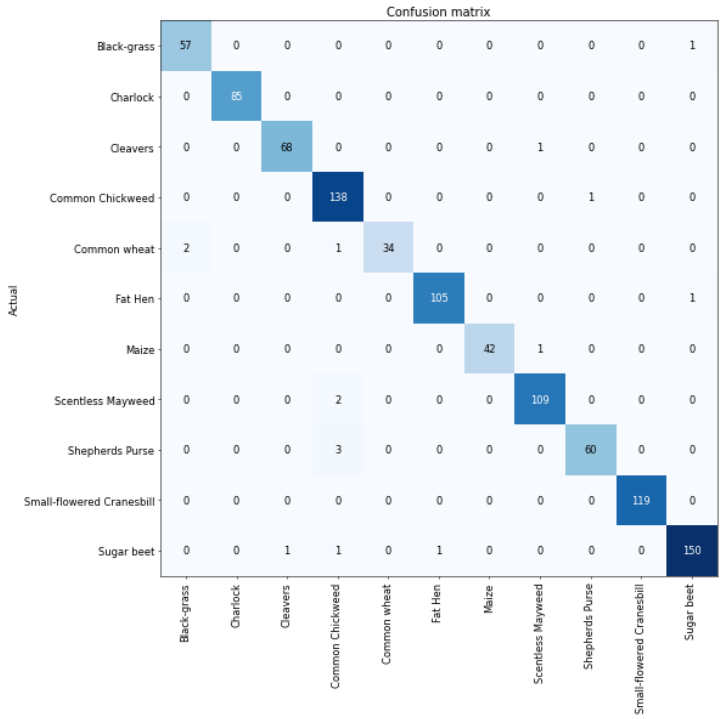

Figure 13: Confusion matrix

Finally, We exported the trained model as a pkl file used for testing, which proved satisfactory.

```
learner.export('plants.pkl')
```

6.8 Testing Result and Discussion

Testing is done by creating a dataset of 900 images that were carefully inspected for inconsistency and inefficiency and then collected in a CSV file, each bearing header number and label. Then we loaded the testing dataset in a separate python file and executed the prediction code line which can be simply written as,

```
test_path = Path("/test/path")
```

```
test_images = get_image_files(test_path)

pred, pred_idx, probs = learn.predict(test_images)
```

7 Conclusion

In this Project, based on the requirements given to us each of the four group members came up with a use case. Those use cases were turned into a use case diagram. From the use case diagram, we created four different tapaal models. All these models well join together in one main unit so we could verify and validate that the system works as intended. Moving on from there we created a deep learning model and initially train it with a plant seedlings data set from Keaggle.We then created a second data to train again and make it more efficient.In this whole process we gain some knowledge on how AI and deep learning can have a huge a positive impact in our daily life.We have also learn how to fine tune the different parameters of a neural network to improve the accuracy of the prediction.There is still room for improvement regarding future work,more especially in creating more dataset. This will help to create agricultural drones that have the capability to identify plants faster and correctly.

8 Contribution of each member in the paper

1. Abstract - Izuchukwu George Enekwa
2. Introduction - Nnaemeka Valentine Eze
3. System Analysis - Nnaemeka Valentine Eze and Jires Donfack Voufo
4. Activity Diagram- Nnaemeka Valentine Eze
5. Use-cases - Nnaemeka Valentine Eze
6. System model - Izuchukwu George Enekwa
7. Verification - Izuchukwu George Enekwa
8. Deep Learning Language Processing - Abdullah Al Forkan
9. Conclusion - Jires Donfack Voufo

Github Repository of our Project:
https://github.com/Forkan01/Autonomous-Systems-B-Lab$_{G}roup - D - [GitHubAutonomousSystemsBLabGroupD]$

References

[1] tapaal.net: Introduction.

[2] Sameh ElGhany, Mai Ibraheem, Madallah Alruwaili, and Mohammed Elmogy. Diagnosis of various skin cancer lesions based on fine-tuned ResNet50 deep network. 68(1):117–135. Publisher: Tech Science Press.

[3] E. Fantin Irudaya Raj, M. Appadurai, and K. Athiappan. Precision farming in modern agriculture. In Amitava Choudhury, Arindam Biswas, T. P. Singh, and Santanu Kumar Ghosh, editors, *Smart Agriculture Automation Using Advanced Technologies: Data Analytics and Machine Learning, Cloud Architecture, Automation and IoT*, Transactions on Computer Systems and Networks, pages 61–87. Springer.

[4] Pooja Sharma. The future of indian agriculture.