Robotics Application in Precision Spraying

Puspa Kamal Poudel

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University in
partial fulfillment of the requirements for the degree of

Master of Science
in
Crop, Soil and Environmental Science

Hasan Seyyedhasani, Chair
Mizuho Nita
Sherif M. Sherif

February 6, 2024
Blacksburg, Virginia

Keywords: robotics, grape cluster mapping, precision spraying, planning

Robotics Application in Precision Spraying

Puspa Kamal Poudel

Abstract

This thesis presents an investigation on innovative approaches to agricultural management, addressing challenges in both viticulture and turfgrass management. The first topic of this thesis introduces the Adaptive Crop Load Estimation (ACLE) method, a deep learning-based grape counting approach designed to alleviate the need for extensive annotated datasets. By training the model on a limited set of images, this method demonstrates promising results in accurately estimating grape cluster counts across different zones in the vineyards, with an average Mean Absolute Error (MAE)/Root Mean Square Error (RMSE) of 0.86/0.66. The ACLE method aims to reduce the cost of deploying automated grape counting systems by minimizing manual image annotation efforts and enabling model reusability across different vineyards.

The second topic of this thesis delves into the realm of Turfgrass management, recognizing its pivotal roles in environmental health and aesthetics. Focusing on the challenges posed by spot-based diseases, the study introduces the Spot Treatment Pathfinding and Scheduling (STPAS) method. This framework employs Unmanned Ground Vehicles (UGV) for targeted spot spraying, optimizing robot stops and trajectories based on varying scenarios such as different spot sizes and robot capabilities. The trajectory planner developed within STPAS utilizes GPS coordinates and the radius of affected areas to determine efficient stops and paths for autonomous vehicles. Comparative analysis on the developed simulators reveals that STPAS reduces the distance traveled and time taken for spot spraying by over 50% compared to

conventional boom-based sprayers, thereby enhancing both economic and environmental sustainability in Turfgrass management practices.

Robotics Application in Precision Spraying

Puspa Kamal Poudel

General Audience Abstract

This thesis explores solutions for improving agricultural practices, specifically focusing on grapevine cultivation and turfgrass management. The first part introduces a novel method called Adaptive Crop Load Estimation (ACLE), which employs deep learning to accurately count grape clusters in vineyards. Unlike traditional methods requiring extensive annotated data, ACLE demonstrates significant results with minimal training images, aiming to reduce the cost of automated grape counting systems and enhance their adaptability across various vineyards.

In the second part, the thesis delves into development of planning algorithm for precision spot spraying. Addressing challenges posed by spot-based diseases, the study introduces the Spot Treatment Pathfinding and Scheduling (STPAS) method. This algorithm provides robot stops and optimizes routes based on different scenarios such as spot sizes and robot capabilities. Comparative analysis of the simulation results reveals that STPAS improves efficiency, reducing both the distance traveled and time taken for spot spraying compared to boom-based sprayers. This not only benefits economic considerations but also contributes to environmental sustainability in turfgrass management practices.

Acknowledgments

I would like to express my gratitude to my advisor, Dr. Hasan Seyyedhasani, for his continuous help and guidance with the projects over the course of my graduate school. I am thankful to my committee members, Dr. Mizuho Nita, and Dr. Sherif M. Sherif, for their help and direction with the thesis project. I am grateful for all the members of the iACT Lab at Corporate Research Center, Virginia Tech who helped me directly or indirectly during my stay. I would not have been able to accomplish this without their input. I would like to thank my loving wife for her unwavering support, understanding, and encouragement throughout my journey of completing this master's thesis. Lastly, my parents and my siblings, who have always been there by my side in the worst and the best, will always hold a special place in my heart.

Table of Contents

List of Figures

List of Tables

Chapter 1 Crop load-based zoning of grape vineyard using deep learning models.

Abstract

Grape cluster count is a critical metric in developing crop-load management to ensure crop quality and quantity. Growers rely on their prior experience and/or manual count in sample areas/vines to estimate the count of grape clusters in vineyards. There has been significant work in grape cluster counting by object detection and instance segmentation techniques. However, all these methods require a huge, annotated dataset. In this work, we propose a simplified, yet robust deep learning-based grape counting method named Adaptive Crop Load Estimation (ACLE) and investigate its performance in vineyard images. Our approach involves training a deep learning model with only eight images to count grape clusters in the rest of the images. Experiments were conducted to acquire images using an unmanned ground vehicle integrated with a vision sensor in the vineyard. Results showed an average Mean Absolute Error (MAE)/Root Mean Square Error (RMSE) of 0.86/0.66 when using multiple batches of eight images to count grapes in the rest of the images. The results are promising in reducing the cost of deployment of automated methods for grape counting by reducing manual image annotation and reusing the model trained on other vineyards.

1.1 Introduction

Viticulture, the cultivation of grapes, is labor-intensive. Many tasks, such as identifying disease through visual inspection, are time-consuming due to the size of vineyards and the varying attributes of parcels. One of the important tasks that requires manual labor is yield estimation, which is necessary for organizing the yield and maintaining quality standards. A common method for estimating yield is using random grape bunch sampling, which involves assessing

factors such as the number of bunches per vine, the number of berries per bunch, and the weight of the berries (Clingeleffer et al., 2001). However, this approach has limitations, as it is destructive, and there is considerable variation of crop load within the same vineyard, which can be as much as a ten-hold difference.

One potential solution to viticulture tasks' labor-intensive and time-consuming nature is using computer vision algorithms to automate the process. Automatically detecting and counting grape bunches using computer vision would enable large-scale processing and more accurate modeling. The use of computer vision for grape detection was first suggested by Dunn et al. to assess the potential of computer-vision-based yield estimation (Dunn & Martin, 2004).

Recently, machine learning techniques incorporating vision-based modeling have been widely used for fruit detection and counting using proximal imaging in various cropping systems. These techniques have been applied to orchard crops such as apples (Bargoti & Underwood, 2017), oranges (Maldonado & Barbosa, 2016), and mangos (Payne et al., 2013), as well as specialty crops like tomatoes (Mu et al., 2020, Rahnemoonfar and Sheppard, 2017) and grapes (Cruz et al., 2019; Liu et al., 2017; T. T. Santos et al., 2020). In viticulture, most of the existing research on yield estimation using proximal imagery has focused on using feature engineering and computer vision to count individual grape berries (Millan, 2018; Nuske, 2014), although some studies have also examined the relationship between pixel count and grapevine yield (Diago et al., 2012).

These techniques generally used a traditional approach to computer vision, involving the selection of detection algorithms, feature extractors, and classifiers based on subjective criteria. However, developing specific algorithms for this purpose can be tedious and time-consuming, as it often involves image calibration using artificial lighting or backgrounds and high-quality cameras.

An alternative approach that has gained popularity in recent years is deep learning with convolutional neural networks (CNNs). These models utilize multiple stacked layers, including convolutional, pooling, and fully connected layers, to perform both feature extraction and classification (or segmentation). Deep learning allows for end-to-end learning, which can simplify complex algorithms by learning robust feature representations. Since a CNN model won the ImageNet Large Scale Visual Recognition Challenge in 2012, deep learning has become the leading approach for image classification (Deng, 2009; Krizhevsky, 2012).

End-to-end learning has been successfully demonstrated for grape yield estimation using proximal imagery, in which a regression CNN model was used to directly predict yield in mass units from images (Silver and Monga, 2019). However, the images in that study were collected individually using a smartphone, which is not a practical solution for scaling up the process. Additionally, the vines were prepared specifically for imaging with a calibration marker in each frame, and the images required extensive manual processing and cropping. Despite these limitations, the study demonstrates the potential of using deep learning for viticultural yield estimation.

Olenskyj et al compared the performance of a transformer model and an end-to-end regression model for estimating grape yield, using 107,933 images to train both models. The mean absolute error was 18% and 18.5% for the transformer and end-to-end regression model, respectively (Olenskyj et al., 2022). Santos et al. also used a CNN model to identify grapes and obtained an F1 score of up to 0.91 for instance segmentation, using a total of 300 images for the training (T. T. Santos et al., 2020). Bargoti S et al. employed a deep learning model to identify apples in orchards and achieved the best pixelwise F1-score of 0.79 (Bargoti & Underwood, 2017).

Mohimont et al used a deep learning model with a UNet architecture and achieved an IoU score of up to 0.76 and a counting accuracy of 86% (Mohimont, 2021).

The works previously mentioned all used large, annotated datasets. The annotations are cumbersome and costly. We propose a new approach for determining the number of grape clusters in rows of grapevines by utilizing less annotated data, which we refer to as Adaptive Crop Load Estimation (ACLE). This method involves using an unmanned ground vehicle called Husky, which has a GoPro camera sensor. First, the camera system takes continuous lateral images of the canopy as the robot moves along the rows. Few images from a single row are annotated and a pre-trained deep learning model is fine-tuned with these images. This trained model is then used to identify and count the number of grape clusters in the rest of the vineyard.

To achieve this, we used a publicly available vineyard dataset to train a deep learning model. This trained model serves as the starting point for fine-tuning when a new vineyard needs to be analyzed. The fine-tuning process enables the model to adapt to the specific characteristics of the new vineyard, thus improving its accuracy in estimating crop load. To fine-tune the model, we acquire a small number of images of a single row of the new vineyard and use them to update the parameters of the pre-trained model. The fine-tuned model is then used to predict the crop load in the entire new grape vineyard. The images of the vineyard are acquired by a camera system mounted on an unmanned ground vehicle.

We tested the effectiveness of this approach by using two popular deep learning-based image segmentation models, UNet and Multi-Scale Attention Network (MANet). UNet and MANet are both encoder and decoder-based architectures, except MANet has an additional component called Pixel Attention Block (PAB) in its decoder. To further improve the performance of these models, we modified these networks by substituting their feature-extracting layers with pre-

trained backbones from Residual Network (ResNet). Due to its simplicity and performance, the ResNet network has been a popular choice for feature extractors in agriculture application.

Our A-CLE approach offers a foundation for a simple and efficient method of estimating crop load in a vineyard, requiring only a small amount of labeled data from the new vineyard. It can be easily adaptable to new vineyards without having to train the deep learning model from scratch. By synergizing with autonomous ground vehicles, this methodology is suitable for large-scale crop yield forecasting and precision agriculture management. The objectives of this work are:

Objective 1: crop load estimation of a vineyard using deep learning models.

Objective 2: Identification of zones which require inspections.

1.2 Materials and Methods

The proposed approach is based on utilizing a combination of an unmanned ground vehicle and a deep learning-based image segmentation model to estimate the grape cluster count distribution in a vineyard.

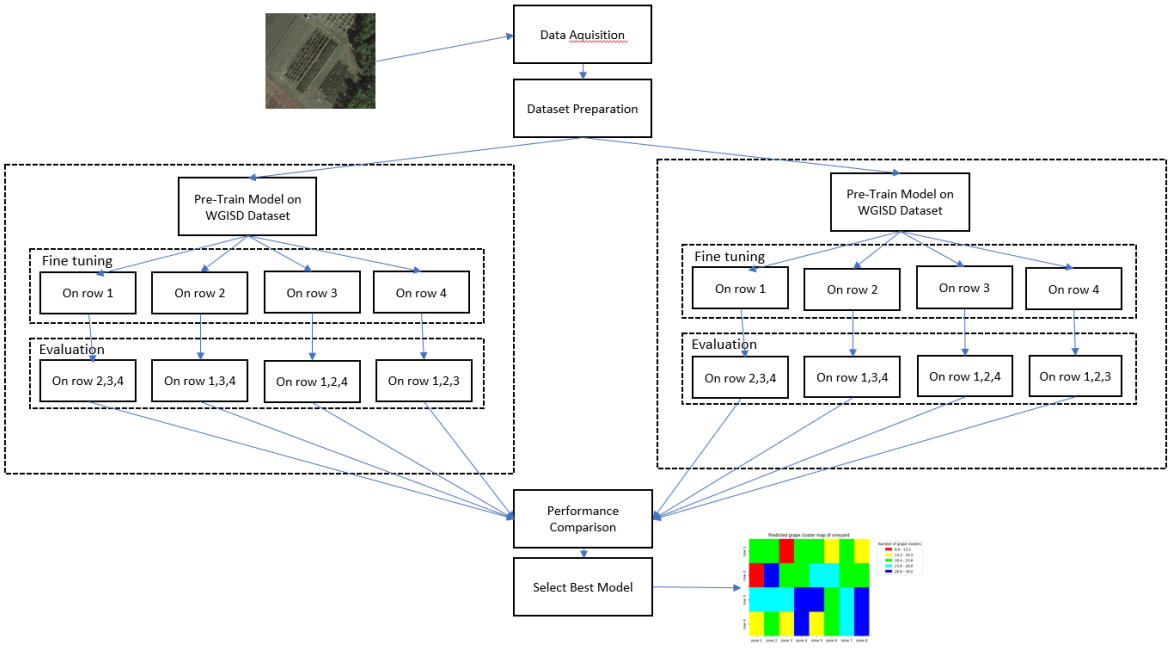


Figure 1 Overview of this work.

Data collection

Embrapa Wine Grape Instance Segmentation Dataset (WGISD)

We used the Embrapa wine grape instance segmentation dataset (T. Santos et al., 2019) as a publicly available grape vineyard dataset for the A-CLE approach. The images were taken at a vineyard in Espírito Santo do Pinhal, São Paulo, Brazil. The images were captured in April 2017 for Syrah and April 2018 for other varieties (Chardonnay, Cabernet Franc, Cabernet Sauvignon and Sauvignon Blanc). A Canon EOS REBEL T3i DSLR camera and a Motorola Z2 Play smartphone were used to take the images, which were captured at distances of around 1-2 meters from the vines. With the EOS REBEL T3i, they took 240 images of all cultivars, and with the Z2 smartphone, they took 60 images covering all cultivars except Syrah. The REBEL and Z2 images were scaled to 2,048 X 1,365 pixels and 2,048 X 1,536 pixels, respectively. Out of these 300 images, 137 images had annotations with masks. The dataset already includes suggested train

and test splits. The training set includes 110 images, and the testing set includes 27 images. The annotated masks representing the location of grape pixels in the image were also present in the dataset. We combined multiple masks to create a single mask of grapes for our purpose.

Figure 2 Sample images in the WGISD vine image dataset.

AREC

We collected images from a vineyard at Alson H. Smith AREC (Agricultural Research and Extension Center, Winchester, VA) using a GoPro (HERO10 Black, GoPro Inc, Los Angeles, USA) camera mounted on an unmanned ground vehicle (Husky, Clearpath robotics, Canada). We collected eight images per row for four rows (Figure 3). The latitude and longitude of the field was 39°06'34.4"N 78°16'57.0"W. Altogether, we collected 32 high-quality images. Each image represents a zone in each row. That is, each row is divided into eight zones. Our goal is to use images from a single row to identify and count grape clusters in other rows.

Figure 3 Dataset acquisition site. The length of a single row was 85m and each row was divided into 8 zones at equal distance apart.

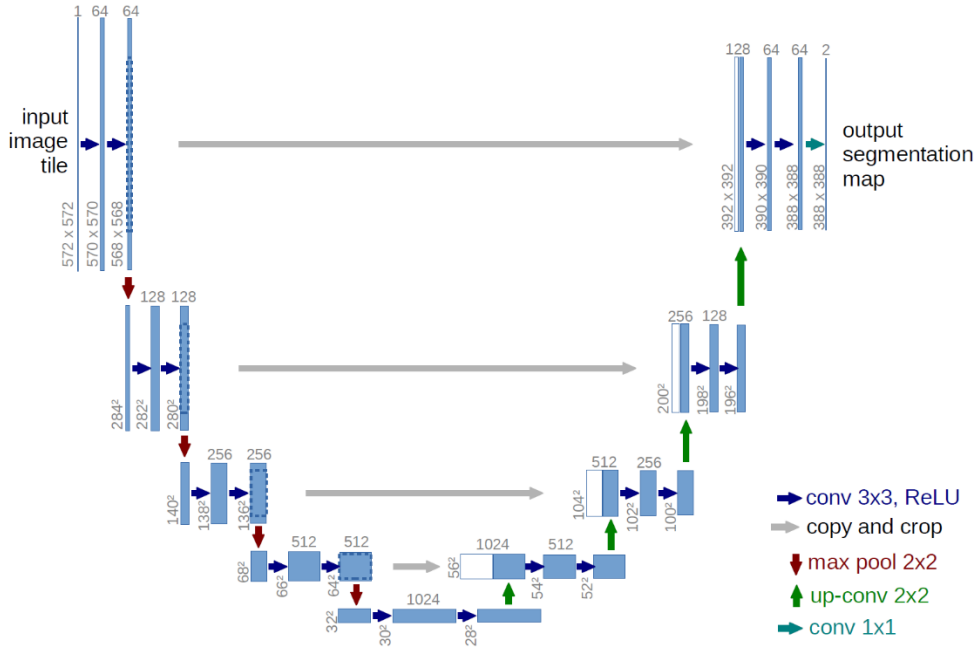
Figure 4 Images captured by Husky UGV

Segmentation Network

We tested our approach using two commonly used lightweight semantic segmentation models, i.e., UNet (Ronneberger et al., 2015) and Multiscale attention network (MANet) (Fan et al., 2020).

UNet

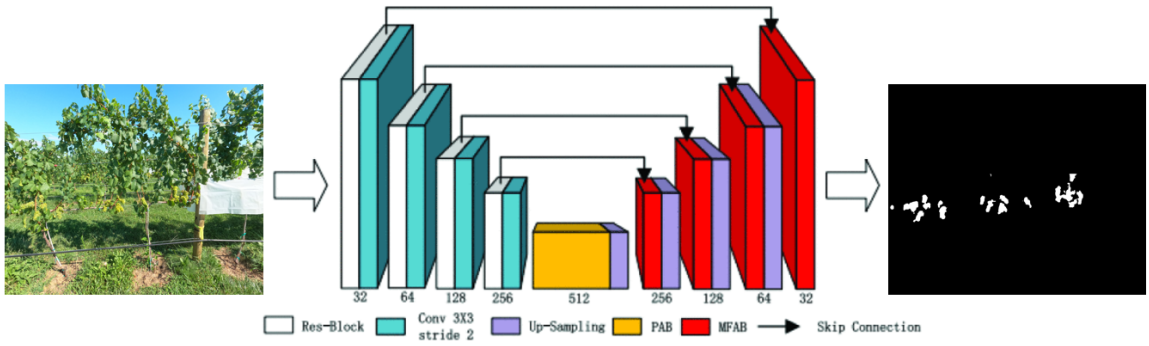
UNet is a deep learning architecture that was developed for the task of medical image segmentation (Ronneberger et al., 2015). It is a fully convolutional neural network that was designed to perform pixel-wise classification tasks, such as identifying different objects or abnormalities in images.



The UNet architecture consists of an encoder and a decoder, which are connected by a series of skip connections. The encoder consists of a series of convolutional and pooling layers that reduce the spatial resolution of the input image, while the decoder consists of a series of up sampling and convolutional layers that increase the spatial resolution of the output. The encoder acts as a feature-extracting network while the decoder utilizes the feature extracted to generate an output mask representing the class each pixel of the input's belongings to. The skip connections allow the decoder to access low-level features from the encoder, which helps to improve the accuracy of the model. UNet has been widely used in a variety of medical image analysis tasks, including the segmentation of organs, tissues, and tumors. It has also been applied to other domains, such as satellite image analysis and industrial inspection (Augustaukas & Lipnickas, 2019; de Jong & Bosman, 2019).

Multi-scale Attention Network

The multi-scale attention network (MA-Net) (Fan et al., 2020) was first used for the segmentation of livers and tumors. This network is unique compared to a traditional U-Net due to its incorporation of two attention-based modules. The first of these is the Position-wise Attention Block (PAB), which captures the relationships between different feature maps. The architecture of the MA-Net is depicted below.



The authors of the MA-Net utilized PAB to model a wide range of spatial contextual information across local feature maps. They also introduced the Multi-scale Fusion Attention Block (MFAB) inside the decoder block, which determines the relative importance of individual feature maps from multiple feature maps and enhances the relevant feature map while suppressing those that are less relevant to the segmentation task.

We hypothesize that the MFAB module would have a similar impact on grape cluster segmentation, as it would enhance feature maps that capture the geometric shapes of crop rows in images.

Feature Extracting Backbones

Feature extraction backbones are commonly used in deep learning networks such as UNet because they provide a pre-trained set of features that can be fine-tuned for a specific task. These backbones are typically pre-trained on large datasets and can extract high-level features, such as edges, textures, and shapes, from input images. By using these pre-trained features as the starting point, UNet reduces the amount of training required to perform well on a specific task while also improving the overall accuracy of the network.

Moreover, feature extraction backbones help address the overfitting problem, where the network memorizes the training data instead of generalizing it to unseen examples. By using pre-trained features, the network can focus on learning task-specific features rather than learning low-level features from scratch, which can reduce overfitting. We chose to use the ResNet-34 backbone because of its simplicity and wide use in the field of agriculture.

Resnet as backbone

ResNet-34 is a deep network with 34 layers, including convolutional, batch normalization, and activation layers. The network can extract high-level features from input images, such as edges, textures, and shapes, through multiple convolutional operations. The network can also be fine-tuned for specific tasks, such as image classification, semantic segmentation, or object detection, by using the pre-trained features as a starting point. (He et al., 2015)

The network is built on the concept of residual connections, where the input to a layer is added to the output of the same layer. This allows the network to learn the residual mapping between the input and the output, making it easier for the gradients to flow through the network during training. This model is also called resnet34 in deep learning libraries and thus we use resnet34 to refer to this model in this work.

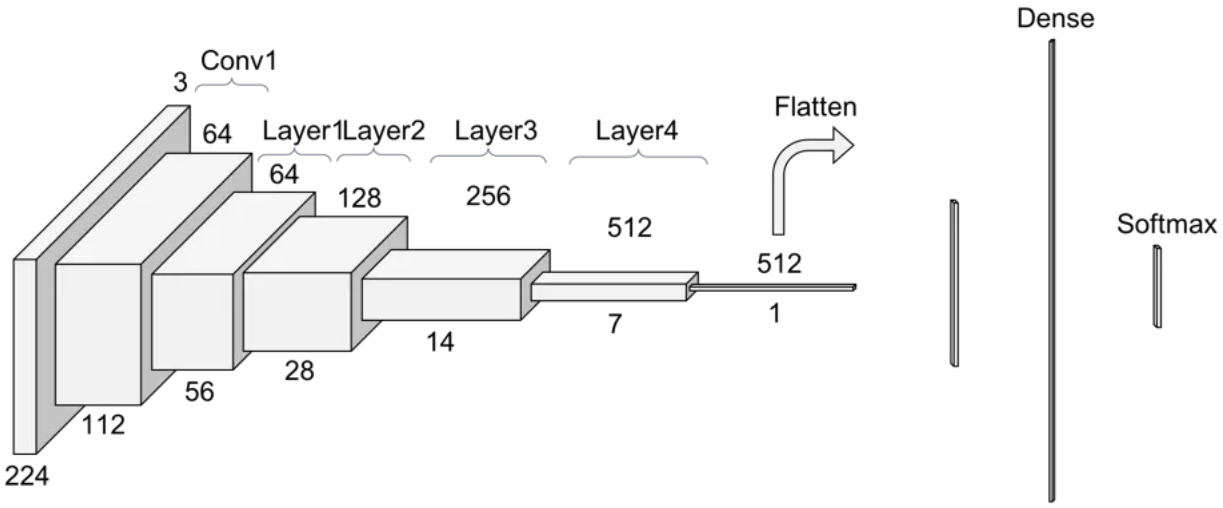


Figure 5 The architecture of ResNet-34 model. *We used features extracted by Conv1, layer1, layer2, and layer3 to extract the necessary features to feed into the decoder module.*

Modified UNet and MANet

We modified UNet and MANet architectures by changing their feature-extracting networks to resnet backbone.

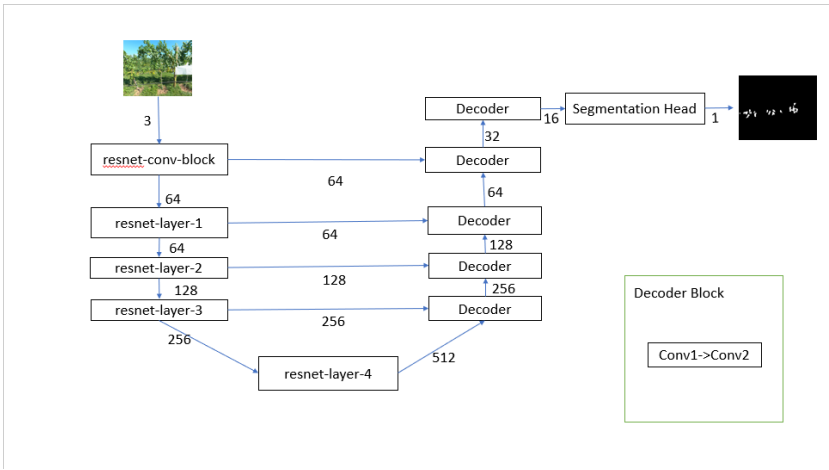


Figure 6 Modified UNet with resnet34 backbone

Figure 6 is the modified UNet which includes feature extracted from resnet34 architecture. The resnet-conv-block and resnet-layer-x are the layers that extract the required features from the input image of the channels. These layers are the part of resnet34 architecture. These features are

then fed to the decoder block, which aggregates the features and increases the spatial resolution using convolution operation. Finally, the segmentation head utilizes single-point convolution to generate an output mask representing the detected grapes in the image.

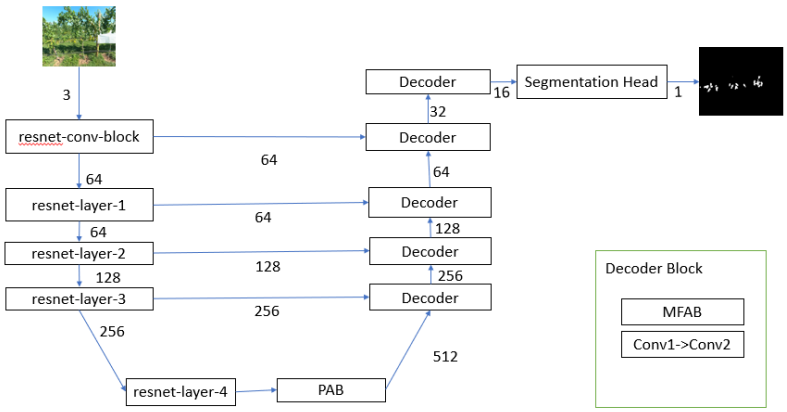


Figure 7 Modified MANet with resnet34 backbone. The decoder block in MANet is different than UNet because it includes an additional MFAB module.

Figure 7 shows the modified manet architecture where the feature extracting blocks are replaced by the blocks from the pre-trained resnet34 networks. This architecture is similar to the UNet architecture except that it includes a Position Attention Block (PAB), which aggregates rich contextual information over local feature maps according to the spatial attention map and considers the long-range spatial dependency between features in a global view.

Additionally, the decoder block has an additional MFAB block, which extracts the interdependence among feature channels by combining the High and Low-level feature maps.

Loss Function and Optimizer

Binary Cross entropy loss function

Binary cross-entropy loss, also known as log loss, is a commonly used loss function in binary classification tasks. Given a set of predicted probabilities for a binary outcome (e.g., 0 or 1) and a set of true labels, the binary cross-entropy loss is defined as:

$$Loss = -(1/N) * \sum[y * log(p) + (1 - y) * log(1 - p)] \quad (1)$$

Where N is the number of examples, y is the true label (either 0 or 1), p is the predicted probability of the positive class (e.g., the probability that y is 1)

The binary cross-entropy loss is a measure of the difference between the predicted probabilities and the true labels. It penalizes the model more for predictions that are far from the true labels. For example, if the true label is 1 and the predicted probability is 0, the loss will be very high because the prediction is very far from the true label. On the other hand, if the true label is 0 and the predicted probability is also 0, the loss will be low because the prediction is close to the true label.

Dice loss

Dice loss is a loss function commonly used in image segmentation tasks, where the goal is to predict a binary mask for each pixel in an image. Given a set of predicted masks and a set of true masks, the Dice loss is defined as:

$$Loss = 1 - \left(2 * \frac{\sum(y * p)}{\sum(y^2) + \sum(p^2)}\right) \quad (2)$$

Where y is the true mask (either 0 or 1), and p is the predicted mask (also either 0 or 1).

The Dice loss is a measure of the overlap between the predicted masks and the true masks. It is calculated by first summing the element-wise product of the true masks and the predicted masks and dividing this by the sum of the squares of the true masks and the predicted masks.

The Dice loss ranges from 0 to 1, with a value of 0 indicating perfect overlap between the true masks and the predicted masks and a value of 1 indicating no overlap. The loss is minimized when the predicted masks are similar to the true masks.

Dice loss is often used in combination with other loss functions, such as cross-entropy loss, to improve the performance of image segmentation models. It is particularly useful for tasks where the proportion of positive and negative pixels in the masks is imbalanced, as it can provide a more balanced measure of performance compared to other loss functions. In our approach, we used the sum of cross-entropy loss and dice loss as a loss function to tune the parameters of the model.

Adam optimizer

Adam (Adaptive Moment Estimation) is a widely used optimization algorithm for training neural networks. It is an extension of the popular stochastic gradient descent (SGD) algorithm and is particularly well-suited for training deep learning models. The Adam algorithm uses a combination of the gradient of the parameters with the moving average of the past gradients to adapt the learning rate. The main advantage of Adam over SGD is that it can automatically adjust the learning rate for each parameter based on the historical gradient information.

The Adam algorithm uses the following update rule for each parameter:

$$\theta_{t+1} = \theta_t - \alpha * \sqrt{\left(\frac{m_t}{v + \epsilon}\right)} * g_t \quad (3)$$

Where, θ_{t+1} is the updated parameter, θ_t is the current parameter, α is the learning rate, m_t is the moving average of the gradient at time t, v_t is the moving average of the squared gradient at time t, ϵ is a small constant value added to the denominator to prevent division by zero and g_t is the gradient of the parameter at time t.

The moving averages m_t and v_t are calculated as follows:

where:

$$m_t = \beta_1 * m_t - 1 + (1 - \beta_1) * g_t \quad (4)$$

$$v_t = \beta_2 * v_t - 1 + (1 - \beta_2) * g_t^2 \quad (5)$$

β_1 and β_2 are the decay rates for the moving averages, typically set to 0.9 and 0.999 respectively.

The Adam optimizer also includes a bias correction term for the moving averages m_t and v_t, which helps to correct the bias in the early iterations of the optimization.

The Adam optimizer typically requires less fine-tuning of the learning rate compared to other optimization algorithms such as SGD and is a popular choice for training deep learning models.

Network Trainings

Training on the WGISD dataset

The images in the WGISD dataset were of high resolution (2,000 by 1,365 pixels). Deep learning models like UNet and MANet work well with image input of resolution in the multiple of 32. The images in the training set and validation set were split into blocks of smaller resolution (512 by 512 pixels). Hence, 1,324 image blocks were generated from 110 training images, and 324 image blocks were generated from 27 testing images. This corresponds to 1,320 image blocks for the training set and 324 image blocks for the validation set. The training process was performed on an RTX A4000 GPU (NVIDIA, Santa Clara, CA) for a total of 200 epochs, with a batch size of 16. The model was trained with a custom loss function, the sum of binary cross entropy loss and dice loss. Similarly, Adam was used as an optimizer to update the parameter weights according to the loss function. The training results were evaluated in terms of the dice coefficient.

After the training process, only the model with parameters that had the best dice coefficient was saved in the pth file format.

Table 1 Dataset distribution, loss function and optimizer used for pretraining.

No. of training images	1,320
No. of validation images	320
Loss function	Binary cross entropy loss + dice loss
Optimizer	Adam

Transfer learning on our dataset

Due to the high resolution of the acquired images, they were split into blocks of resolution (1024 by 1392 pixels). Since all the images were taken from approximately equal distances from the vines, the images were subjected to the image processing technique depicted below:

The raw image was first cropped to the resolution (5,120 by 1,392 pixels). Then, five blocks of resolution (1,024 by 1,392 pixels) were generated for each image. Hence, 40 blocks were generated from 8 images per vineyard row. From these images, 30 images were selected as a training set and 10 images were selected as a validation set for tuning learning rate and speed up the convergence. The transfer learning process was performed on the same dedicated computing system (i.e., NVIDIA RTX A4000 GPU) for a total of 200 epochs, with a batch size of 16. The model performs well for the input of resolution (512 by 512 pixels), so the random crop data augmentation technique was applied during the training process. Since only 40 images are used for training purposes, the data augmentation technique also makes sure the model is not overfitted during the training process. Additionally, to prevent overfitting, only the model weight with the highest validation score during the training process was selected for further evaluation on the unseen dataset.

Figure 8 Five small images are generated from one main image.

The custom loss function, which is the sum of binary cross entropy loss and dice loss, was used as a loss function and Adam as an optimizer. In order to evaluate the model properly, the model is often evaluated on the unseen data from a similar distribution. We evaluated the models with the images of rows different from the row the model was trained on. For instance, if the model was fine-tuned on images from row-1, it was evaluated with images from row-2, row-3, and row-4. Thus, we have 40 images for finetuning the model and 120 images for evaluation of the model.

Identifying and counting grapes.

After the model was trained on a single row of the vineyard, the images from the rest of the rows were fed to the model for identification and segmentation of grape clusters. The model outputs the mask representing detected grape clusters in the images. Since the images from other rows are of high resolution, a detection pipeline was created, as depicted below.

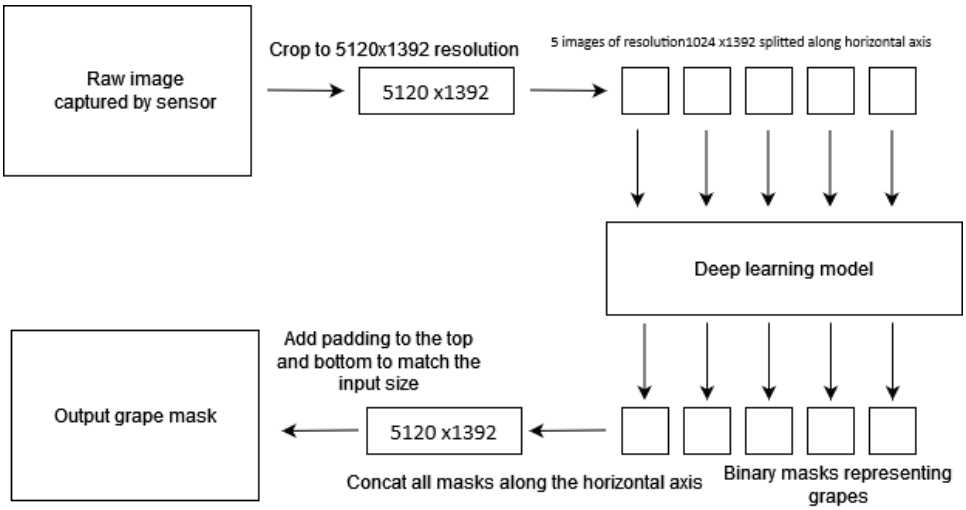


Figure 9 Grape cluster detection pipeline: A high-resolution image is split into five images, and their detections are merged back to generate the output grape mask.

The pipeline involves splitting each high-resolution image into five blocks using the technique explained in section 2.6.2 and then feeding these blocks to the model, giving five output masks. The output masks are combined in a similar fashion to form a single output mask of the same resolution as the original image. The output mask is compared to the ground truth mask, and IoU scores and pixel-based recall scores are calculated.

Additionally, the clusters of grapes were also estimated using the output mask from the model. To estimate clusters from the grape mask, it is first necessary to find out how many pixels in the original image are roughly equal to one cluster of grapes. Since most of the grapes in our dataset are close to each other, identifying clusters using contour detection in the mask would yield impractical results. Therefore, we used approximation from the ground truth data to calculate the number of pixels in one cluster of grapes instead of applying contour detection. To do that, we estimated the average grape cluster pixel number (or pixels per single cluster, *ppsc*) from ten manually counted cluster images.

Using the calculated *ppsc*, the number of clusters of grapes in each zone in each row was counted using a ground truth mask and a predicted mask.

$$C = Number\ of\ pixels\ identified\ as\ grapes\ in\ the\ mask\ /\ ppsc \quad (6)$$

Performance Evaluation

We investigated the ACLE approach with two commonly used lightweight segmentation models, i.e., UNet and MANet. To evaluate the performance of each model for the ACLE, we utilized IoU score, mean absolute error (MAE), root mean squared error (RMSE), and R2 score metrics.

IoU score

The Intersection over Union (IoU) score, also known as the Jaccard Index, is a common evaluation metric used in image segmentations. It quantifies the accuracy of an object detector on a particular dataset by measuring the overlap between the predicted segmentation and the ground truth. Mathematically, the IoU score is calculated as the area of overlap between the predicted segmentation (A) and the ground truth (B) divided by the union area of these two regions. The formula for computing the IoU is:

$$IoU(A, B) = \frac{Area\ of\ A \cup B}{Area\ of\ A \cap B} \quad (7)$$

The IoU score considers class imbalance issue usually present in image segmentation problem. For example, if the model predicts every pixel of an image to be background, the IoU measure

can effectively penalize for that, as the intersection between the predicted and ground-truth regions would be zero, thereby producing an IoU count of 0.

This metric ranges from 0 to 1, where 0 indicates no overlap, and 1 signifies perfect overlap. A higher IoU score means the segmentation model is more accurate in delineating the target object from the background or other objects. It's especially useful in scenarios where precise object localization is crucial, such as medical image analysis, autonomous driving, and satellite image interpretation. The IoU score is robust as it penalizes both false positives (area outside the ground truth but inside the predicted segmentation) and false negatives (area inside the ground truth but outside the predicted segmentation).

Root Mean Square Error

The Root Mean Squared Error (RMSE) is a widely used metric for evaluating the performance of the algorithm in terms of the accuracy of the grape cluster count. It measures the difference between the predicted cluster count and the true cluster count. The RMSE is calculated by taking the square root of the mean of the squared differences between the predicted and actual cluster counts. Mathematically, it can be defined as follows:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\mathrm{p_i} - y_i)^2}{n}} \tag{8}$$

Where p is the predicted cluster count, y is the true cluster count, n is the total number of data rows of which cluster count is estimated. The RMSE metric provides an aggregate measure of the deviation of the predicted cluster count from the true cluster count. It ranges from 0 to infinity, and a low value of RMSE indicates better performance of the clustering algorithm.

Mean absolute error

The Mean Absolute Error (MAE) is another widely used metric for evaluating the performance of algorithms in terms of the accuracy of the grape cluster count. Like the Root Mean Squared Error (RMSE), it is a measure of the difference between the predicted cluster count and the true cluster count. The MAE metric is the average absolute difference between the predicted and true clusters counts. The MAE can be mathematically defined as follows:

$$MAE = \sqrt{\sum_{i=1}^{n} \frac{p_i - y_i}{n}} \tag{9}$$

p is the predicted cluster count, y is the true cluster count, n is the total number of data rows of which cluster count is estimated. Like RMSE, the MAE metric also provides an aggregate measure of the deviation of the predicted cluster count from the true cluster count. It ranges from 0 to infinity, and a low value of MAE indicates better performance of the clustering algorithm. MAE is less sensitive to outliers than RMSE because it doesn't square the difference between the predicted and true value, is easy to interpret, and is more robust to outliers than the RMSE.

R2 score

The R-squared (R^2) score, also known as the coefficient of determination, is a statistical measure widely used in regression analysis to assess the goodness of fit of a model. It represents the proportion of the variance in the dependent variable that is predictable from the independent variables. In the context of grape cluster counting, R2 score indicates how well the counting of grape clusters through model output matches the true counts of grape clusters.

Recall Score

Recall, also known as sensitivity or true positive rate, measures the ability of an image segmentation model to identify pixels of the target object correctly. A high recall score indicates that the model effectively captures most of the relevant pixels, minimizing the risk of missing important regions.

1.3 Results and Discussion

Model Pre-Training (Training on WGISD Dataset)

Error! Reference source not found. compares performance metrics between the UNet Model and the MANet Model for grape cluster segmentation, after we trained them with the WGISD dataset. The UNet model, with an IoU score of 84%, demonstrates similar performance to the MANet model. Also, the recall metric for models differs only 1%, i.e., 92% and 93% for UNet and MANet models, respectively. However, the parameter size for the UNet model is smaller than that of the MANet model, by nearly 23%. This makes the UNet model advantageous for its deployment on edge-based devices. Since a model with fewer parameters enhance speed and efficiency relative to models with larger parameter sizes. Therefore, the UNet model appears to

be a more optimal choice for precise grape cluster segmentation, facilitating the generation of an accurate crop load map for vineyards.

Table 2 Evaluation metrics of the models trained with WGISD dataset, where the IoU indicates a model's capability to segment grape clusters accurately, and the recall score indicates the percentage of grape pixels accurately identified.

Model Type	IoU Score	Recall Score	Parameter size (MB)
UNet Model with Resnet34 backbone	84%	92%	24.44M
MANet Model with Resnet34 backbone	84%	93%	31.78M

Model Adaption to a Vineyard

Error! Reference source not found. shows the selected model's performance in identifying grapes on the images collected from the studied vineyard. The Rows column corresponds to each row of the vineyard on which a model was trained and tested on the remaining rows of the vineyard. As with training with the pre-training stage, the IoU Score and Recall Score columns show the quantified metrics when the model was evaluated.

Table 3. Comparison of performance of the UNet-Resnet34 and MANet-Resnet34 models, on identifying grape pixels when trained only with one row of the studied vineyard.

	Rows	IoU Score	Recall Score
A-CLE with U-Net-ResNet34	1	82%	91%
	2	83%	90%

	3	82%	91%
	4	82%	89%
	Avg	82%	90%
A-CLE with MANet-ResNet34	1	81%	91%
	2	82%	90%
	3	80%	91%
	4	78%	85%
	Avg	80%	89%

It can be observed that the model when trained on different rows exhibits different scores. Hence, the average score was used to evaluate the performance of each adopted model using the ACLE approach. As demonstrated in Table 3, both models, when trained on a single vineyard row, were able to identify grapes in the remaining vineyard rows with similar accuracy. The UNet-Resnet34 model outperformed the MANet-Resnet34 model based on IoU and recall scores. The average IoU score was achieved at 82% with the UNet-Resnet34 model and 80% with the MANet-Resnet34 model. This indicates that the UNet-Resnet34 model was able to generate a grape segmentation mask of 82% of the actual segmentation mask. And as the average recall score shows, the UNet-Resnet34 model correctly identified 90% of grape pixels (Table 3).

The results, of adapting the models, showed that the performance metrics have improved when applied to the studied vineyard compared to applying the WGISD dataset for pre-training (shown in Table 2). This was expected since the WGISD dataset is a grape dataset from a different

vineyard on which we first trained the model; then, by applying the model to our studied vineyard, the model learned more specific features of the vineyard by fine-tuning its weight to the new vineyard.

Grape Cluster Estimation

After the training process, we employed the model, trained on respective rows, to estimate the number of grape clusters in the remaining rows of the vineyard. Since individual grape clusters were not clearly differentiated in the captured images, we formulated the problem as a semantic segmentation problem of the identification of grape pixels in the image. In order to get the grape count from the mask representing grape pixels, we divided the total number of grape pixels by the approximate number of grape pixels per grape cluster. Figure 10 shows the distribution of the number of pixels in ten identifiable average-sized grapes in our dataset. We used the mean value of 39,420 pixels as the approximate count of the number of pixels in a single grape cluster. To calculate the RMSE, MAE, and R2 score, the number of pixels classified as grapes was divided by the average number of pixels per grape cluster.

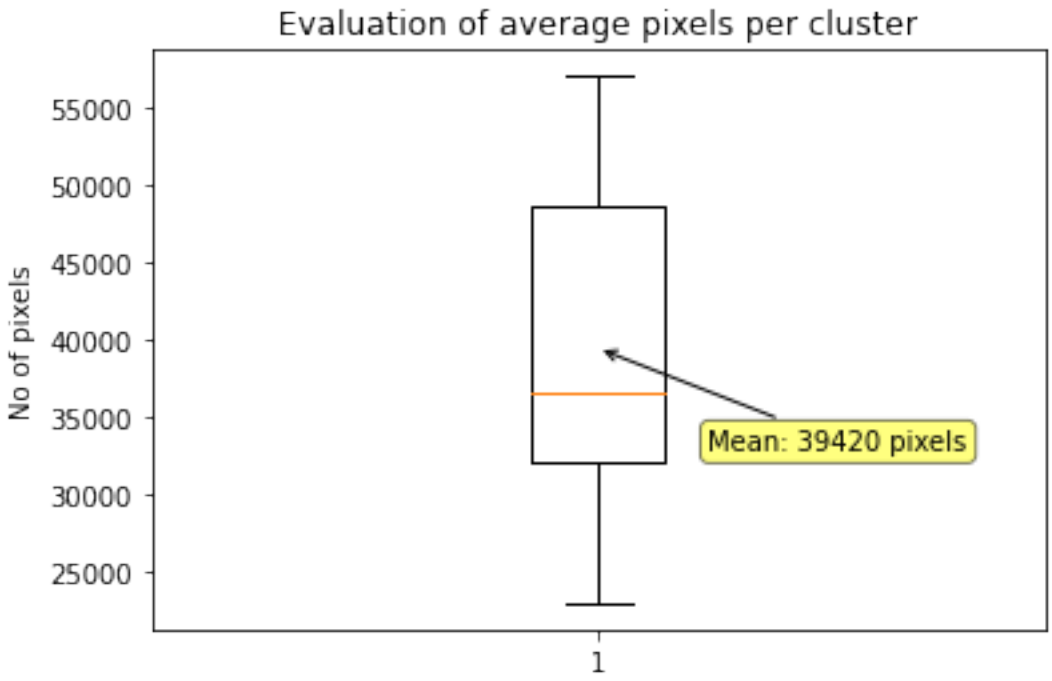


Figure 10 Distribution of the number of pixels per cluster of ten average-sized grapes in our dataset

Table 4 shows the RMSE, MAE, and R2 score of the grape cluster counts as estimated by different models trained on other rows. As with **Error! Reference source not found.**, the Rows column represents each vineyard row used to fine-tune the model; therefore, each model has been evaluated with acquired images from the remaining rows of the vineyard.

Table 4. of the evaluation metrics for the grape cluster estimations using MANet-ResNet34 and UNet-ResNet34 models

	Rows	**RMSE**	**MAE**	**R2**
A-CLE with U-Net-Resnet34	1	1.414	1.083	93.3%
	2	0.866	0.667	97.6%
	3	1.369	1.042	92.5%

	4	1.208	0.875	95.3%
	Avg	1.214	0.917	94.7%
A-CLE with MANet-Resnet34	1	1.5	1.167	92.5%
	2	0.979	0.792	97.0%
	3	1.429	1.125	91.8%
	4	1.871	1.5	88.7%
	Avg	1.445	1.146	92.5%

Table 5 Comparison of UNet and MANet models for grape cluster estimation.

	IoU	**Recall**	**RMSE**	**MAE**	**R2**
UNet-Resnet34	82.0%	90.3%	1.214	0.917	94.7%
MANet-Resnet34	80.2%	89.0%	1.445	1.146	92.5%

From the results shown in Table 5, the UNet-Resnet34 model performs better than MANet-ResNet34 model in all studied metrics, IoU, Recall, RMSE, MAE, and R2. Therefore, we selected the UNet-ResNet34 model for generating a crop load map of our zoned vineyard field.

In our proposed method, we have trained models based on each vineyard row. As presented in Table 6, the selected model, UNet-ResNet34 when trained on row 2 of the vineyard, generated

the best results with RMSE of 0.87 and R^2 of 0.98. Therefore, the UNet-Resnet34 model trained on row 2 is selected as the best model to create the correct crop load map of the vineyard.

Table 6 Comparison of performance of UNet-ResNet34 model trained on different rows. The model trained on row 2 has been better at estimating grape cluster counts in the rest of the rows.

Rows	IoU Score	Recall Score	RMSE	MAE	R2	Remarks
1	82%	90%	1.41	1.08	93%	
2	82%	90%	0.86	0.66	97%	Best
3	81%	91%	1.36	1.04	92%	
4	81%	88%	1.20	0.87	95%	

Crop Load Map Generation

Figure 11 illustrates the crop load map for the vineyard as calculated from the annotated ground truth mask as well as the crop load map for the vineyard as predicted by the best model. It is evident that row 1 and row 4 exhibit lower crop loads in comparison to the remaining rows, and row 3 appears to have relatively the highest crop load.

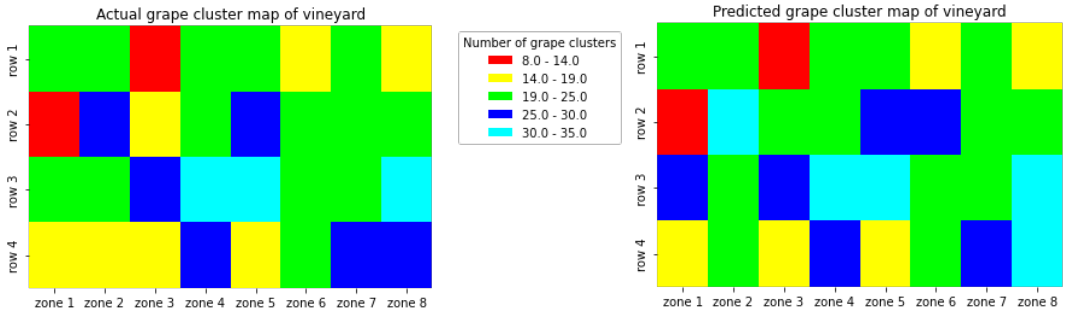


Figure 11 Crop load map for the vineyard as calculated from the annotated ground truth mask and crop load map for the vineyard as estimated by the best-performing model, i.e., UNet-ResNet34 model trained on row 2.

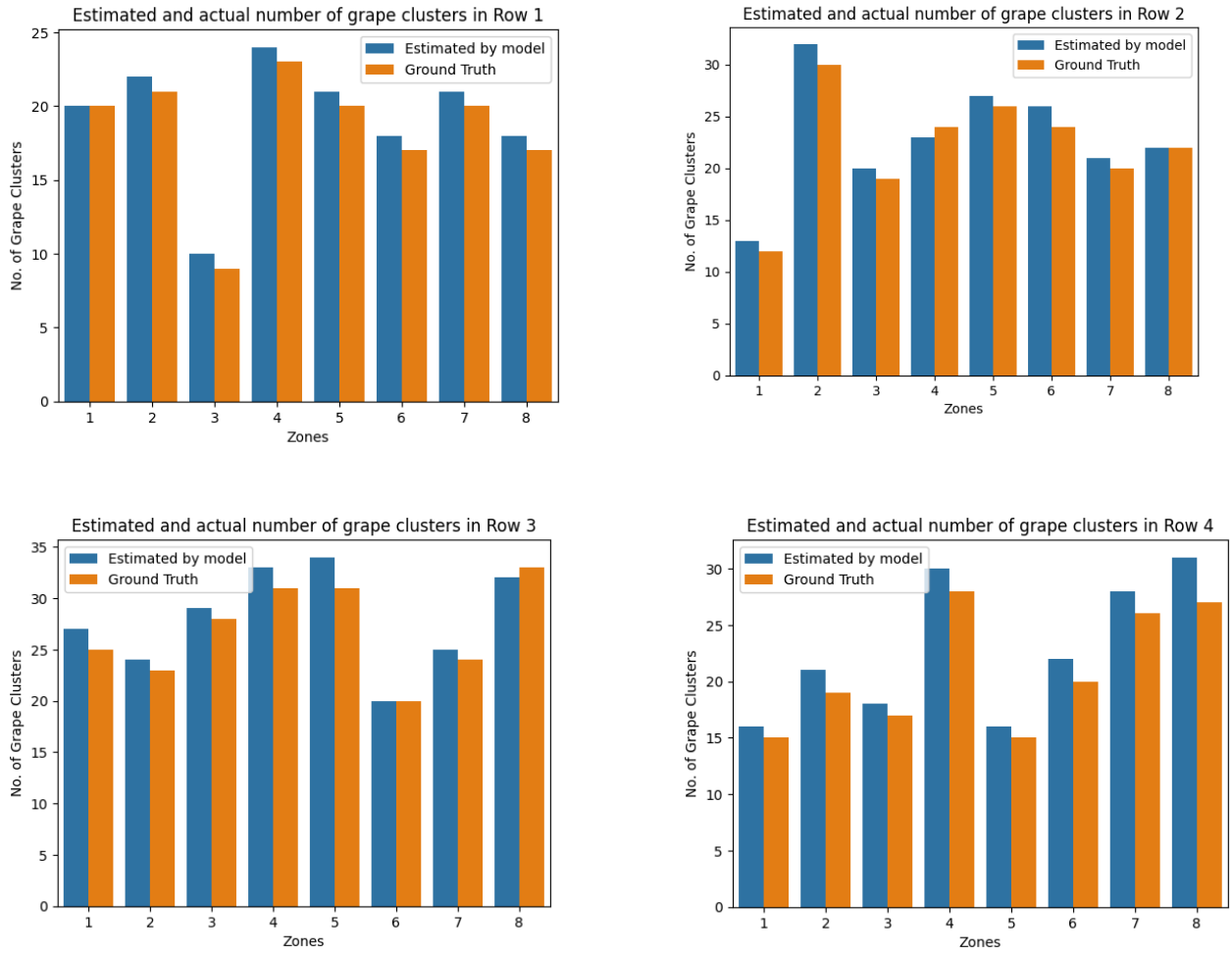


Figure 12 Comparison of estimated and actual number of grape clusters in row 1, row 2, row 3, and row 4.

To classify the zones based on their predicted crop load, we used the best model to predict the number of grape clusters in each zone. Then, we color-coded the predictions in five classes (red, yellow, green, cyan, or blue) to indicate the range of crop load. The red color indicates a low

crop load, while the blue color indicates a high crop load. As illustrated in Figure 11, zone 3 in row 1 and zone 1 in row 2 require immediate attention by a grower to investigate further.

Table 7 Classification metrics based on whether the model was able to accurately classify zones.

Classification Classes	Precision	Recall	Support
Red (8.0 – 14.0)	100%	100%	2
Yellow (14.0 - 19.0)	100%	71%	6
Green (19.0-25.0)	86%	86%	12
Cyan (25.0-30.0)	71%	67%	8
Blue (30.0-35.0)	60%	100%	4
Weighted Average	84%	81%	32

Precision represents the percentage of predictions that are correct. For instance, 86% of the estimated green zones are true. In the same way, recall percentage represents the percentage of the zones that have been accurately estimated to be of given clusters. This is crucially important as among all the zones, the red zone often needs to be inspected. Since the red zone has a recall of 100%, the model accurately and correctly identifies all the zones that require attention. However, only two data points for the red zone might be insufficient to make the conclusion. Overall, the model is 81% accurate in estimating the cluster counts in different zones in the vineyard. More experiments in different vineyards are needed for more accurate insights.

1.4 Potential Future Work

One potential direction for future work is to evaluate the model's performance on a wider range of grape varieties and growing conditions to ensure its generalizability and applicability in different scenarios. Furthermore, it may be beneficial to develop a user-friendly interface for the model, such as a mobile app, to enable easy and intuitive use by grape growers and other stakeholders. Another potential direction for future work is to work on videos instead of images. Finally, it may be interesting to investigate the potential use of this approach for other applications, such as the identification of other types of crops or objects in agricultural settings.

1.5 Conclusions

Using the modified UNet model to identify and count grape clusters using minimum supervised data yielded satisfactory results. The average root mean square error for counting grapes was 0.842. Although there hasn’t been much significant difference in using the UNet or MANet model, the MANet model tends to output many false positive pixels. This can be attributed to human error in the annotation of the data. In conclusion, we demonstrated that the model trained on another vineyard can be reused to identify and count grapes in a new vineyard with as few as eight annotated images. This decreases the cost of deploying a deep-learning pipeline for counting grape clusters by minimizing the cost of data annotation.

1.6 References

Aquino, A., Millan, B., Diago, M. P., & Tardaguila, J. (2018). Automated early yield prediction in vineyards from on-the-go image acquisition. *Comput. Electron. Agric.*, *144*, 26–36. https://doi.org/10.1016/j.compag.2017.11.026

Augustaukas, R., & Lipnickas, A. (2019). Pixel-wise road pavement defects detection using U-net deep neural network. *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, *1*, 468–471.

Bargoti, S., & Underwood, J. (2017a). Deep fruit detection in orchards. *Proceedings - IEEE International Conference on Robotics and Automation*, 3626–3633. https://doi.org/10.1109/ICRA.2017.7989417

Bargoti, S., & Underwood, J. P. (2017b). Image Segmentation for Fruit Detection and Yield Estimation in Apple Orchards. *Journal of Field Robotics*, *34*(6), 1039–1060. https://doi.org/10.1002/ROB.21699

Clingeleffer, P. R., Martin, S. R., Dunn, G. M., & Krstic, M. P. (2001). *Crop development, crop estimation and crop control to secure quality and production of major wine grape varieties: A national approach: Final report to Grape and Wine Research and Development Corporation/principal investigator, Peter Clingeleffer; [prepared and edited by Steve Martin and Gregory Dunn]*.

de Jong, K. L., & Bosman, A. S. (2019). Unsupervised change detection in satellite images using convolutional neural networks. *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Deng, J. (2009). Imagenet: A large-scale hierarchical image database (L. J. Li, Trans.). *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255.

Dunn, G. M., & Martin, S. R. (2004). Yield prediction from digital image analysis: A technique with potential for vineyard assessments prior to harvest. *Aust. J. Grape Wine Res.*, *10*(33), 196–198. https://doi.org/10.1111/j.1755-0238.2004.tb00022.x

Fan, T., Wang, G., Li, Y., & Wang, H. (2020). Ma-net: A multi-scale attention network for liver and tumor segmentation. *IEEE Access*, *8*, 179656–179665.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition* (arXiv:1512.03385). arXiv. https://doi.org/10.48550/arXiv.1512.03385

Krizhevsky, A. (2012). Imagenet classification with deep convolutional neural networks. *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, 1097–1105.

Liu, S., Cossell, S., Tang, J., Dunn, G., & Whitty, M. (2017). A computer vision system for early stage grape yield estimation based on shoot detection. *Computers and Electronics in Agriculture*, *137*, 88–101. https://doi.org/10.1016/j.compag.2017.03.013

Liu, S., Zeng, X., & Whitty, M. (2020). A vision-based robust grape berry counting algorithm for fast calibration-free bunch weight estimation in the field. *Comput. Electron. Agric.*, *173*, 11. https://doi.org/10.1016/j.compag.2020.105360

Maldonado, W., & Barbosa, J. C. (2016). Automatic green fruit counting in orange trees using digital images. *Computers and Electronics in Agriculture*, *127*, 572–581. https://doi.org/10.1016/j.compag.2016.07.023

Millan, B. (2018). Grapevine yield estimation using image analysis and machine learning. In A. Aquino (Ed.), & J. Tardaguila (Trans.), *Journal of Sensors* (Vol. 2018). https://doi.org/10.1155/2018/9634752

Mohimont, L. (2021). Comparison of Machine Learning and Deep Learning Methods for Grape Cluster Segmentation (N. Gaveau, Trans.). *Smart and Sustainable Agriculture*, 64–75.

Montoya-Cavero, L. E., Díaz de León Torres, R., Gómez-Espinosa, A., & Escobedo Cabello, J. A. (2022). Vision systems for harvesting robots: Produce detection and localization. *Computers and Electronics in Agriculture*, *192*, 106562. https://doi.org/10.1016/J.COMPAG.2021.106562

Nuske, S. (2014). Automated visual yield estimation in vineyards (L. Yoder, Trans.). *Journal of Field Robotics*, *31*(55), 837–860.

Olenskyj, A. G., Sams, B. S., Fei, Z., Singh, V., Raja, P. v., Bornhorst, G. M., & Earles, J. M. (2022). End-to-end deep learning for directly estimating grape yield from ground-based imagery. *Computers and Electronics in Agriculture*, *198*, 107081. https://doi.org/10.1016/j.compag.2022.107081

Payne, A. B., Walsh, K. B., Subedi, P. P., & Jarvis, D. (2013). Estimation of mango crop yield using image analysis – Segmentation method. *Computers and Electronics in Agriculture*, *91*, 57–64. https://doi.org/10.1016/j.compag.2012.11.009

Santos, T., de Souza, L., Andreza, dos S., & Avila, S. (2019). *Embrapa Wine Grape Instance Segmentation Dataset – Embrapa WGISD* [dataset]. Zenodo. https://doi.org/10.5281/zenodo.3361736

Santos, T. T., de Souza, L. L., dos Santos, A. A., & Avila, S. (2020). Grape detection, segmentation, and tracking using deep neural networks and three-dimensional association. *Computers and Electronics in Agriculture*, *170*, 105247. https://doi.org/10.1016/J.COMPAG.2020.105247

Chapter 2 Towards Autonomous Spot Fungicide Application: Providing autonomous precision spot spraying solutions with unmanned ground vehicles.

Abstract

Turfgrass, recognized as the third-largest agricultural crop by land area, plays crucial roles beyond aesthetics. It contributes to environmental health by reducing erosion, cooling surroundings, improving soil quality, and purifying the air. However, maintaining these vast grasslands poses significant challenges, particularly in disease and weed management. Traditional methods of turfgrass management often lead to overuse of chemicals, adversely affecting financial and environmental health. This study explores the STPAS (Spot Treatment Pathfinding and Scheduling) method to find optimum robot stops and trajectories for spot fungicide applications. It considers various scenarios, including differing spot sizes, robot capabilities, and complex environments filled with obstacles and varying terrains. Utilizing GPS coordinates and the radius of affected areas, STPAS efficiently determines the stops and paths for robots, reducing distance traveled and time taken for spot spraying by more than 50%.

2.1 Introduction

Grass coverings predominately cover various landscapes, like airports, sports grounds, graveyards, religious establishments, commercial properties, golf courses, residential gardens, educational institutions, parks, and roadsides. In the U.S., turfgrass expansively covers over 8.1 million hectares, encompassing household gardens, golf fields, and numerous other venues. Notably, this makes turfgrass the third-largest agricultural crop by land area across the nation (Breuninger et al., 2015). Beyond aesthetics and recreational purposes, turfgrass has significant environmental roles. Turfgrass helps stop erosion from wind and water and controls floods. It

breaks down organic chemicals, reduces noise, and makes soil healthier by adding organic matter, resulting in a strong, dense lawn. Turfgrass cools the surroundings, especially in hot times, dropping temperatures by 7 to 14 degrees. This cooling can help reduce the need for air conditioning in summer. Turfgrass also cleans the air. It takes in carbon dioxide and gives out oxygen through photosynthesis. For example, a 25-square-foot patch of healthy turf can supply a day's oxygen for one person. Turfgrass also slows down water runoff and takes in extra nutrients.

Turfgrass landscapes are frequently susceptible to a variety of diseases and weed infestations notably spring dead spot and dollar spots. Conventional management often involves treating large sections or the entire area during an outbreak. This approach leads to considerable chemical wastage, with implications for both financial and environmental health. Over-application disrupts the microbial balance in the soil, potentially impacting soil health and broader environmental systems.

The challenges presented by this traditional method have driven the turfgrass industry and research community toward spot spraying. This targeted approach detects and localizes affected areas, delivering treatments solely to those specific spots. The benefits include reduced chemical use, a minimized environmental footprint, and potential cost savings.

Much of the research work has been carried out in the field of detection of critical spots that need treatment. Deep convolutional neural networks have been widely used for weed detection in turfgrass (Yu et al., 2019, Jin et al., 2022). Much of the research work has been carried out in the application of UAVs for spot spraying.

Accurate path planning is essential to maximize the benefits of spot spraying. Both Unmanned Aerial Systems (UAS) and Unmanned Ground Vehicles (UGV) must follow efficient routes to

ensure effectiveness. Recent research has emphasized the development of guidance and control strategies for autonomous systems, especially in precision agriculture applications (Shi et al., 2023). These strategies aim to safely and autonomously treat affected areas, optimizing the route for efficiency and thoroughness.

Effective path planning is vital due to the complexity of the environments these systems operate in, which are full of obstacles and varied terrains. Strategies like the receding horizon approach continuously plan local trajectories, ensuring adherence to vehicle dynamics and avoiding obstacles. Additionally, spot spraying requires the system to accurately localize and target specific areas, from disease hotspots to weed clusters.

We propose the STPAS (Spot Treatment Pathfinding and Scheduling) method for unmanned ground vehicles to perform spot spraying to address these challenges. This method determines optimum robot stops and trajectories for spot spraying and is designed to accommodate various scenarios, including different spot and robot sizes. By using the coordinates and radius of the affected spot, this method first finds the optimum stops for the robot and then the optimum trajectory to traverse these stops.

Objectives:

1. Develop and test the STPAS method to provide solutions for AGV spot spraying.
2. Test the algorithm's feasibility by implementing it on simulation and real robots.

2.2 Materials and Methods

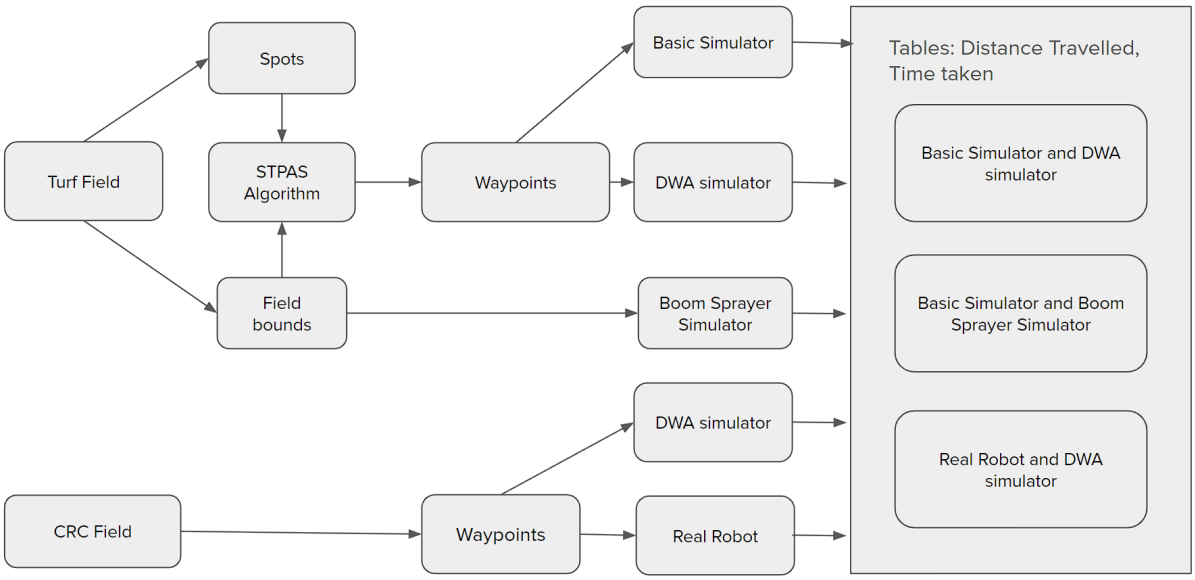


Figure 13 Visual overview of the entire work.

Problem Modelling

In the context of treating specific spots, we define a set S to represent these locations. Each element in this set is characterized by a triplet, represented by $S_i = (x_i, y_i, r_i)$. In this expression, and x_i and y_i denote the UTM coordinates of the spot, and r_i signifies the radius of the spot that needs treatment.

Our robot, tasked with treating these spots, is geometrically represented as a circle. This circle has a defined radius, given by R_s. Moreover, the robot's operational capabilities in terms of treating spots are defined by a specific range. This treatment range is given by R_r

The primary objective of our problem is to ascertain the most efficient stops the robot should make during its operation. This set of optimal stops is represented as $R_{\text{STOPS}} = \{(x_j, y_j)\}$ where x_j and y_j are the UTM coordinates where the robot halts to perform the treatment. The

overarching goal, thereby, is not just to determine these stops but also to find an optimum trajectory that enables the robot to spray all the designated spots in the shortest time feasible. The optimum trajectory, T_{opt}, is defined by a set of robot stop coordinates that minimizes the total distance traveled between all consecutive robot stops in the sequence.

$$T_{opt} = \text{argmin} \sum_{i=1}^{n-1} d\left(R_{\text{STOP}_i}, R_{\text{STOP}_{i+1}}\right) \tag{10}$$

UTM projection

The data for spots often use GPS coordinates for localization. Due to the inherent limitations of the WGS (World Geodetic System) for robotics operations, where working with a cartesian coordinate system is often common, it is important to convert traditional latitudes and longitudes into a more compatible format. The Universal Transverse Mercator (UTM) projection method is a widely used system for representing Earth's locations in a Cartesian manner.

The Universal Transverse Mercator (UTM) system is a global map projection method that divides the Earth's surface into longitudinal zones, each 6 degrees wide, and applies a Transverse Mercator projection to transform the Earth's curved surface into a flat, rectangular grid. Instead of latitude and longitude, UTM uses easting (measured in meters) and northing (also in meters) coordinates to represent locations within each zone, with an origin point specific to each zone. UTM is widely employed in various fields, including robotics, cartography, and navigation, due to its ability to provide accurate and convenient cartesian coordinates, making it suitable for our application.

Solving for the spots

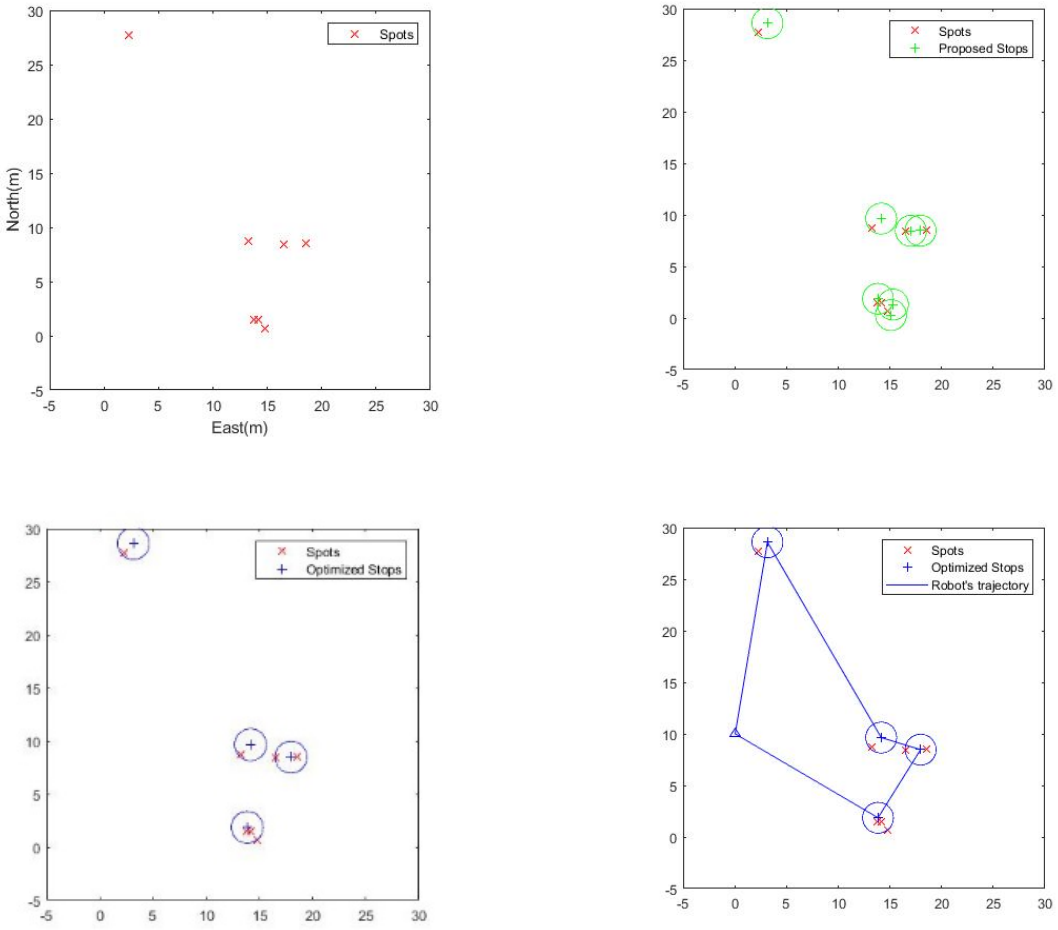


Figure 14 Steps involved in solution generation for spot spraying. First, the proposed stops are generated algebraically. Next, the best stops are selected using a mixed integer linear programming solver. Finally, an optimum trajectory is calculated by solving th.

The process of determining the ideal spots involves three primary phases. Initially, proposals for the most suitable robot stopping points are generated. Once these proposals are available, the problem of obtaining optimal stopping points is addressed using a Mixed Integer Linear Programming (MLP) solver. Following the identification of optimal stopping points, the optimal trajectory that connects these points is ascertained by resolving the Traveling Salesman Problem (TSP).

Proposal generation

Given a collection of n spots, denoted as S, with each location represented by coordinates $s_i = (x_{ci}, y_{ci})$, a set of N proposals for robot stops is calculated. The preliminary task involves calculating and systematically recording the Euclidean distances between each pair of points in a distance matrix termed D_m. Within this matrix, the element $D_m[i, j]$ quantifies the distance between location s_i and location s_j. Following the calculation of the distance matrix, the algorithm proceeds to identify spot locations in the vicinity of each point s_i . specifically, those that fall within a distance less 2*R_r where R_r is the robot's reach range. This selection is based on the geometrical principle that the intersection of two circles, each with radius R_r, occurs only if the distance between their centers is less than 2*R_r. Leveraging this principle, the algorithm pinpoints potential coordinates for circles centered at each location, maintaining a radius of R_r.

The subsequent step involves the algorithm mapping out potential circles for every point and evaluating the inclusion of other points within these circles. For each point s_i , the circle that contains the greatest number of spots is identified as the optimal circle and added to the proposal list.

Determining best stops

After generating proposals for robot stops, we designed an optimization model to determine the best stops for the robot. The model uses two discrete finite sets: S, which represents spots needing spraying, and N, which represents potential robot stopping positions.

Given:

M: Spots needing spraying.

N: Potential robot stopping positions

f_j: Fixed cost for position j. Given as 10 for all j.

$c_{i,j}$: Variable cost for spraying spot I from position j

y_j: Binary variable indicating if position j is an activated stop for the robot.

$x_{i,j}$: Binary variable indicating if spot I is sprayed from position j

$$\min \sum_{j \in N} f_i y_j + \sum_{i \in M} \sum_{j \in N} c_{i,j} x_{i,j} \tag{11}$$

Constraints:

$$\sum_{j \in N} x_{i,j} = 1 \ \forall i \in M$$

$$x_{i,j} \leq y_j \ \forall i \in M, \forall j \in M$$

$$y_j \in \{0,1\} \forall j \in N$$

$$x_{i,j} \in \{0,1\} \ \forall i \in M, \forall j \in N$$

The variable y_j indicates whether position j is activated as a stop for the robot. Simultaneously, the binary variable $x_{\{i,j\}}$ shows if spot i is sprayed from position j. The goal is to minimize the total costs. These costs include the fixed costs of placing the robot and the variable costs of spraying.

Every position j has a fixed cost, f_j, set at 10. This encourages fewer robot stops. The variable costs $c_{\{i,j\}}$ depend on the distance between the robot's position j and spot i. If a spot is within the robot's sprayer range, we calculate the cost using the Euclidean distance. If a spot is out of range, the cost is set at 10^6, making it impractical.

The model has constraints to ensure each spot is sprayed only once, ensuring high-quality work. It also mandates that spraying happens only at chosen robot positions, ensuring logical robot actions. After solving this optimization problem, we identify the best robot stop positions. The subsequent step is to determine the best route for the robot to visit these positions.

Optimum Path Generation

After obtaining the optimum stops for the robot, the next phase involves determining the optimal path for the robot to halt for spot spraying. We modeled this challenge using the well-known traveling salesman problem. The traveling salesman problem seeks to find the shortest possible route that visits a set of cities exactly once and returns to the origin city. It's a classic optimization problem in the field of operations research and has significant implications in logistics and routing.

We designed the following optimization model:

(12)

$$\textbf{Minimize} \quad \sum_{i \in S} \sum_{j \in S} d_{i,j} x_{i,j}$$

$$\textbf{Subject to} \quad \sum_{i \in S} x_{i,j} = 1 \quad \forall j \in S$$

$$\sum_{j \in S} x_{i,j} = 1 \quad \forall i \in S$$

$$\sum_{i,j \in S, i \neq j} x_{i,j} \leq N - 1$$

$$x_{i,j} \in \{0,1\} \quad \forall i,j \in S, i \neq j$$

The set R_s represents the set of robot stops, with its size being N. denoting the number of optimum stops determined for the robot. The parameter $d_{\{i,j\}}$ denotes the distance between stop i and stop j. This distance measure serves as a metric to evaluate the efficiency of the robot's movement between two given stops. The decision variable $x_{\{i,j\}}$ is a binary variable that represents whether two stops are connected with each other. If the robot moves from stop i directly to stop j, $x_{\{i,j\}}$ takes the value of 1; otherwise, it remains 0. By minimizing the combined distances using these variables, the model seeks the most efficient or shortest path for the robot to cover all the stops.

Upon solving this optimization problem, we obtained the shortest path for the robot to traverse the specified stops.

Solving the optimization problems

We used Julia Lang for the programming language, where we used an optimization package IOPT and an optimization solver Gurobi.

Dynamic Window Approach

Once the optimal robot stops and the sequence of stops have been determined, the next critical phase is the implementation of an autonomous navigation system within the robot to execute this plan effectively. The Dynamic Window Approach (DWA) algorithm is widely favored for its simplicity, efficiency, and online operational capability, which is particularly well-suited to the computational constraints of robotic processors.

The DWA algorithm is a real-time motion planning method that considers the robot's dynamics and its environment to make collision-free movements toward a goal (Fox et al., 1997). The core idea of DWA is to limit the search space for velocities to a "dynamic window" that is determined by the robot's current velocity and acceleration limits. Given the robot's kinematics, this approach ensures that the chosen velocities are always feasible.

The algorithm can be broken down into the following steps:

1. Determine the dynamic window: Based on the current velocity (v, w) of the robot and its acceleration limit $(a\_max\ ,\ w\_max)$, calculate the set of admissible velocities that can be reached within a short interval time dt.
2. Evaluate velocities: For each velocity pair within the dynamic window, a cost function is computed that includes: the distance to the goal, the clearance from the obstacles and velocity of the robot.
3. Select optimal velocities: The velocity pair that minimized the cost function would be choses and the command representing this velocity is sent to the robot’s actuators.
4. The robot’s PID controller would send inputs to the motor to reach the given velocity pair.

5. The above four steps are repeated until the robot reaches the desired goal.

The algorithm continuously adapts to changes in the robot's environment. In the context of the development of AGV for spot spraying, this algorithm is optimal algorithm because of the dynamic nature of the agriculture fields. The online nature of the DWA algorithm allows the AGV to rapidly adapt to dynamic changes in the environment, such as unexpected variations in terrain and unexpected obstacles.

Benchmark

Figure 15 Multi Pro 5800 Turf Sprayer

To test the efficacy of our proposed algorithm for precision spot spraying, we compared our system's performance with a conventional boom sprayer. Conventional boom sprayers cover a whole field for spraying. We selected Multi Pro® 5800 Turf Sprayer (The Toro Company, Minnesota) for benchmarking purposes because it is commonly used in Turfgrass spraying. Due to logistics issues, we could not run the actual sprayer on the field. Hence, we decided to use a numerical approach to calculate the time it would take for the Toro sprayer (Figure 15) to cover

the whole field. We also calculated the distance the sprayer would travel from the numerical simulation of the Turo sprayer. Given a field approximated as a polygon, we calculated the path the sprayer would take to cover the polygon. We then calculated the length of the path and divided it by the average speed of the sprayer. From the literature, we found that the optimum average speed of a sprayer is 1.1 m/s. We divided the length of the path by this value to get the least possible time it would take for a sprayer to cover the whole field. Since turning is often at the speed lower than operation speed when the sprayer is moving in the straight line, we also account for turning speed in our calculation.

Experiment Design

Field Selection

We relied on a simulation environment to test our STPAS algorithm for providing solutions for precision spot spraying. We selected two fields for our project. A field at the Corporate Research Center (37.200543, -80.411128) was selected to test the performance of the developed simulator with the real robot. A field at Turfgrass research center (37.216278, -80.410962) was selected for testing the performance of the proposed method of providing solutions to spot spraying. The spots were generated inside the field using the Python programming language.

Figure 16 The field at Turfgrass Research Center, Blacksburg, Virginia

Figure 17 The field at Corporate Research Center, Blacksburg, Virginia

Spot data generation

To test different scenarios of spot treatment, we utilized randomly generated spot data to test our developed algorithm. We used fixed seed values to make sure our experiments were reproducible. We collected the boundary of the field we were using and fed the boundary

through our developed spot-generation algorithm to generate different spots for treatment. To acquire the boundary of the fields, we employed the GPS RTK system, which comprises two crucial components: a stationary base station and a mobile rover, each configured according to their respective roles. We used two Emlid GPS units to configure each of them as the base station and rover.

Initially, we positioned the base station at a fixed location within the field. The base station then determined its approximate position by averaging GPS data collected over a 5-second window. Once this location approximation was established, it was configured to broadcast correction signals to the rover via a LoRa module. The rover, equipped with GPS data from the satellites and the correction signals from the base station, calculated its precise position with centimeter-level accuracy.

With the GPS RTK system successfully configured, we proceeded to mark the boundary of the field that we were using. After deciding on the boundary of the field, we generated spot data based on an algorithm. The algorithm follows a series of mathematical and geometric steps to ensure that the spots are generated within the given boundary and adhere to certain constraints.

Initially, the algorithm set N as the total number of spots to generate and defined *robot_reach* as the maximum distance a robot can cover, fixed at 1.5 meters. The boundary's coordinates were then converted to the Universal Transverse Mercator (UTM) format. This conversion ensured consistent units, simplifying calculations. The boundary was represented as a polygon *P*, defined by its vertices $V=\{ v_1, v_2, v_3 \ldots . v_n\}$, where each vertex v_i corresponded to a coordinate pair (x_i , y_i).

The core of the algorithm was a loop that continued generating spots until the specified number N had been achieved. Within this loop, there were two primary strategies for positioning these

spots. With a 30% chance, the algorithm generated a spot around an existing spot. The algorithm first selected an existing spot; then, the algorithm calculated a new spot within the robot's reach by randomly determining an angle θ in the range $[0,2\pi)$ and a distance d within $[0, robot\_reach]$. The coordinates of the new (x, y) were found using the trigonometric functions sine and cosine, factoring in the angle and distance.

When the algorithm did not generate a spot around the existing spot, the algorithm randomly generated a spot within the given boundary. The random x and y coordinates were chosen based on the range of the x and y coordinates of P, respectively. Then, the algorithm checks whether the generated spot is within the boundary polygon P. If the spot was at outside of this boundary, it was discarded, which prompted the algorithm to create a new one. After completion, the algorithm outputted N number of spots. We set a fixed seed for random number generation to maintain reproducibility.

Figure 18 100, 200 and 400 spots generated in the turfgrass field.

Robot Specification

A Husky Unmanned Ground Vehicle (UGV) was selected as the primary robot for running and testing our STPAS system of spot spraying. This vehicle was equipped with the Piksi Multi SwiftNav GPS-RTK system to enable GPS-based localization. Additionally, the Clearpath IMU7 sensor was employed to determine the robot's orientation relative to the Earth's frame. Data from the IMU sensor, GPS system, and the robot's own odometry were integrated using an extended Kalman filter, which ensured precise localization of the robot in the Earth's frame.

The stops optimized by the proposed solver were input into a custom ROS package. This package made use of the move_base package to relay movement goals to the robot. The robot then executed its movement goals by using a dynamic window approach, which allowed for real-time motion planning and obstacle avoidance. This approach ensured that the robot could navigate efficiently and safely to the predetermined stops for effective spot spraying.

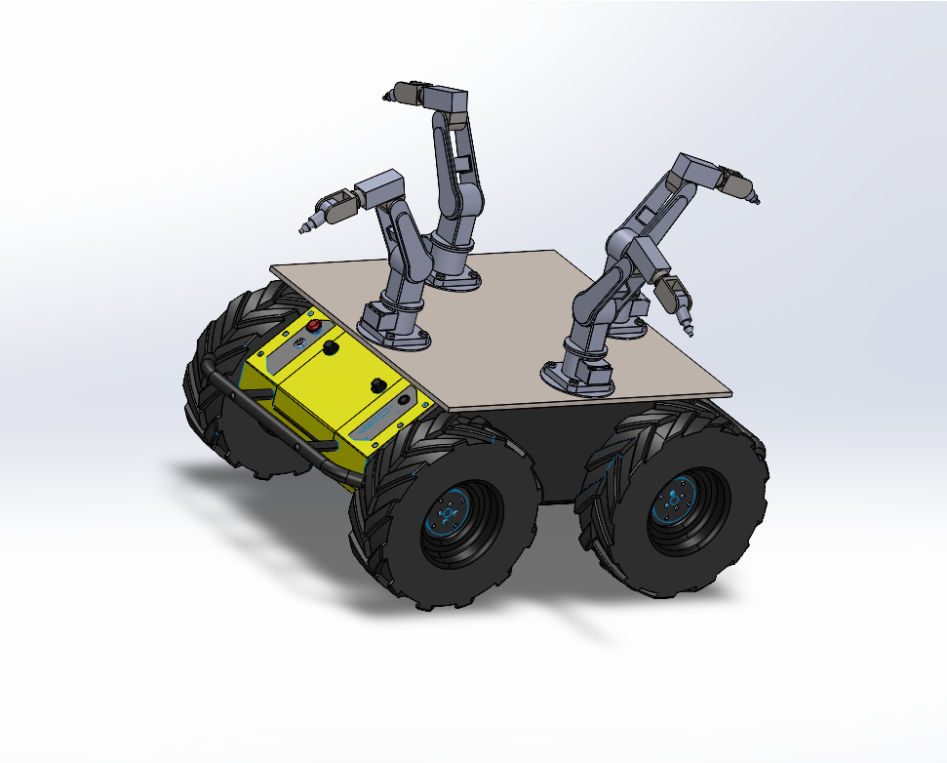


Figure 19 The concept of the robot planned for use in autonomous spot spraying

Simulation

Since it was logistically difficult to always carry the robot around for testing and trial runs. We also developed simulator named DWA_SIM in MATLAB environment to test and evaluate the performance of Autonomous Ground Vehicles (AGVs) when utilizing our innovative Spot Treatment Pathfinding and Scheduling (STPAS) method. The primary objective of this simulator was to provide a robust and accurate assessment of the STPAS method's efficacy in optimizing spot spraying tasks, particularly in comparison to traditional boom sprayers. To ensure the validity and reliability of our simulation results, our initial phase of testing focused on benchmarking the simulator's outputs against actual performance data from the real Husky Robot. Such validation was essential to establish the simulator as a credible tool for evaluating AGV behavior and decision-making algorithms. Upon establishing the credibility of the simulation environment, we performed a comprehensive series of tests to evaluate the STPAS method under various operational scenarios. Mainly we tested scenarios where there are different numbers of spots to treat. The STPAS method was pitted against conventional boom sprayer techniques, allowing us to compare the efficiency between these two approaches. We compared the time taken and distance that the sprayer would take to complete spot spraying task.

2.3 Results and discussion

Boom sprayer

We developed a MATLAB based boom sprayer simulator which simulates the operation of boom-based sprayer to cover the given field. We utilized specification of the Toro multi sprayer (Figure 15) to calculate the distance travelled and time it would take to cover the given field.

Table 8 Specification of Turo sprayers used for benchmark calculation.

Operation width	6 m
Operation speed	1.1 m/s
Turning speed	0.5 m/s
Turning radius	3.2 m

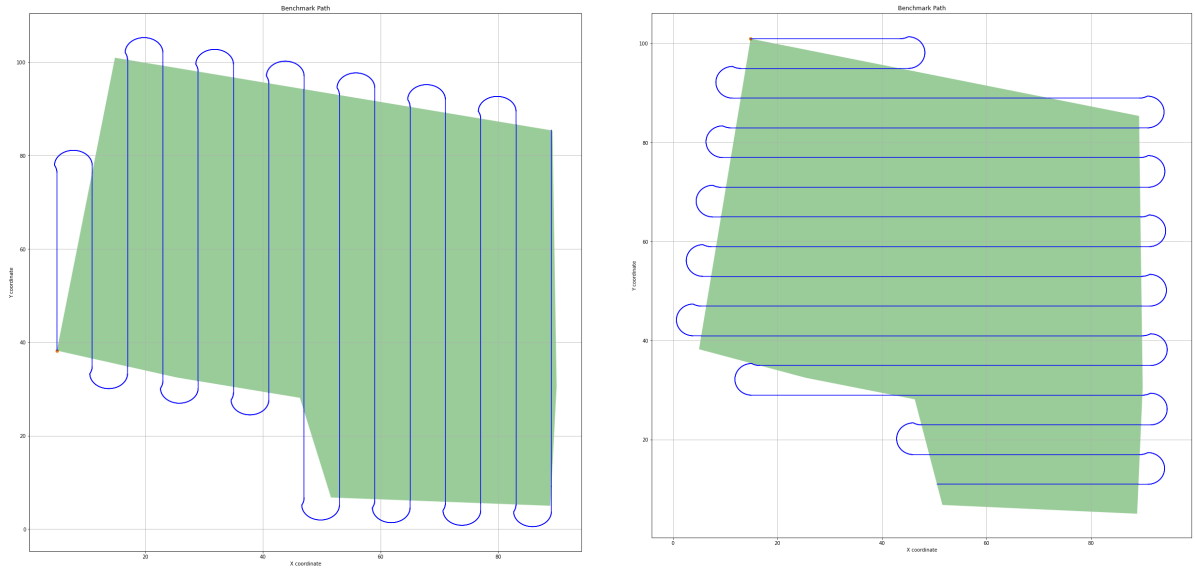


Figure 20 Possible coverage paths followed by Toro Sprayer when treating the field.

Table 9 Distance travelled, and time taken by Turo sprayer to cover the field for spraying propose. In the reference field, moving along east saved time and resulted in less distance travelled.

Coverage pattern	Distance travelled	Time taken	Area covered (acre)

Horizontal (along east)	1150.50 m	20.6 min	0.77
Vertical (along north)	1181.01	20.9 min	0.77

Error! Reference source not found. shows the estimated time taken and distance travelled by Turo sprayer to cover the area of 0.77 acre. The average speed of the Toro Sprayer is assumed to be 2.5mph (1.1 m/s). This is the best minimum estimate because the distance travelled, and time taken also depends on the efficiency of the operator operating the sprayer.

STPAS solution

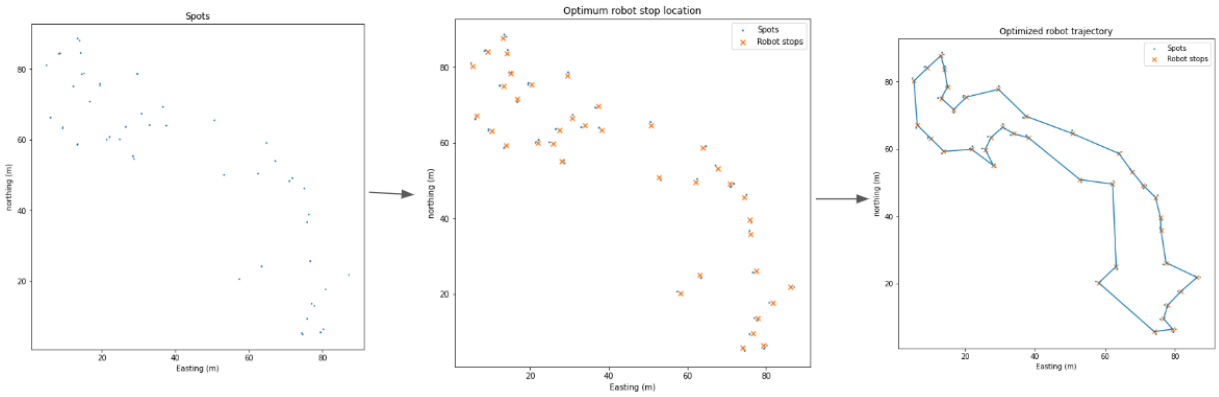


Figure 21 Optimum robot stops and trajectory generation for spot treatment via multi spot application.

Table 10 Time taken for solving for optimum robot stops and trajectories for autonomous spot treatment. The computation time increases exponentially with the number of spots.

No of spots	Solution time
100	12.7 s
200	55.5 s

400	659.5 s

Figure 21 demonstrates the steps with which our solver solves for the optimum stop position and their trajectory. **Error! Reference source not found.**shows the computation time of generating solution for the spots. The total time it takes to solve the optimum position and trajectory depends on the number of spots to process.

Simulator's performance

In order to test the ability for the simulator to closely resemble the actual robot's performance, we performed an experiment where both real robot and simulated robot were provided with the set of waypoints to traverse. The time taken and distance travelled was recorded and compared.

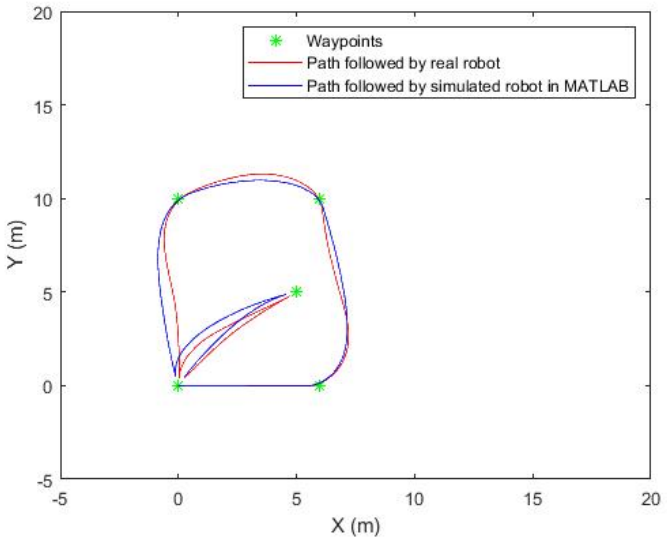


Figure 22 Path followed by real robot and simulated robot while traversing provided waypoints.

Table 11 Time taken by simulated robot and real robot to traverse the given waypoints.

Waypoints	**Real Robot (s)**			**Simulation (s)**	**Error (%)**
	Trial 1	Trial 2	Average		

6,0	8	8	8.00	8.4	5.00
6,10	15	14	14.50	13.4	-7.59
0,10	10	11	10.50	9.3	-11.43
0,0	14	13	13.50	13.1	-2.96
5,5	9	9	9.00	9.8	8.89
0,0	9	9	9.00	9.2	2.22
Total time(s)	65	64	64.50	63.2	-2.02

.

.

Table 12 Distance travelled by simulated robot and real robot to traverse the given waypoint. Ideal distance represents the straight-line distance between the consequent waypoints.

Waypoints	**Ideal distance (m)**	**Real Robot (m)**			**Simulation (m)**	**Error (%)**
		Trial 1	Trial 2	Average		
6,0	6	5.41	5.44	5.43	5.45	0.46
6,10	10	10.37	10.19	10.28	10.35	0.68
0,10	6	6.58	6.63	6.61	6.39	-3.26
0,0	10	10.00	10.02	10.01	10.21	2.00
5,5	5	6.44	6.27	6.36	6.82	7.32
0,0	5	6.06	6.04	6.05	6.26	3.47
Total distance(m)	42	44.86	44.59	44.73	45.48	1.69

We compared the performance of a simulated robotic agent against that of an actual robot by providing both entities with identical sets of waypoints to traverse. The time taken and distance traveled were recorded for both the simulation and real-world executions to gauge the fidelity of the simulation model.

Error! Reference source not found. illustrates the time taken by both the simulated and actual robots to navigate to the given waypoints. The waypoints are defined by coordinates, with the first number representing the x-axis and the second number the y-axis on a two-dimensional

plane. The simulation times are generally close to those observed in the real robot trials, labeled Trial 1 and Trial 2. The total times demonstrate that the simulator is capable of closely approximating the actual robots' behavior, with a mere deviation of -1.9 seconds in Trial 1 and -0.7 seconds in Trial 2 when compared to the simulation total of 63.2 seconds. This indicates a high level of accuracy in the temporal aspects of the simulation.

In **Error! Reference source not found.**, we analyze the distances traveled by the simulated and real robots. The 'Ideal distance' column presents the theoretical minimum distances to travel straight to each waypoint. The simulation and real robot trials (Trial 1 and Trial 2) show slight deviations from these ideal distances, which is expected in practical navigation scenarios due to factors like robot movement dynamics and sensor's measurements not accounted for in the idealized model. The total distances traveled by the simulation and real robots are within a close range of each other and the ideal distance, again illustrating the simulator's high degree of realism in performance.

STPAS method's performance

After confirming that the results reported by the simulator is close to the actual results as reported by the robot. We ran different scenario of the precision spot spraying in the simulator we developed. We also calculated the performance of an ideal robot when the STPAS algorithm is utilized.

Table 13 Comparison of performance of developed simulator for spraying 100, 200 and 400 spots.

	Distance Travelled (m)	**Time Taken (min)**

	100 spots	200 spots	400 spots	100 spots	200 spots	400 spots
STPAS (Ideal Robot)	513.81	728.33	983.51	8.6	12.2	16.4
STPAS (Simulated Robot)	525.67	749.87	1027.30	13.8	22.1	34.1
Error (%)	2.31	2.96	4.45	60.47	81.15	107.93

Table 14 Comparison of robot travel distance and times using STPAS solution against travel metrics of boom sprayer.

	Distance Travelled (m)			**Time Taken (min)**		
	100 spots	200 spots	400 spots	100 spots	200 spots	400 spots
STPAS (Ideal Robot)	513.81	728.33	983.51	8.6	12.2	16.4
Boom sprayer	1150.50	1150.50	1150.50	20.6	20.6	20.6
Improvement (%)	55.34	36.69	14.51	58.25	40.78	20.39

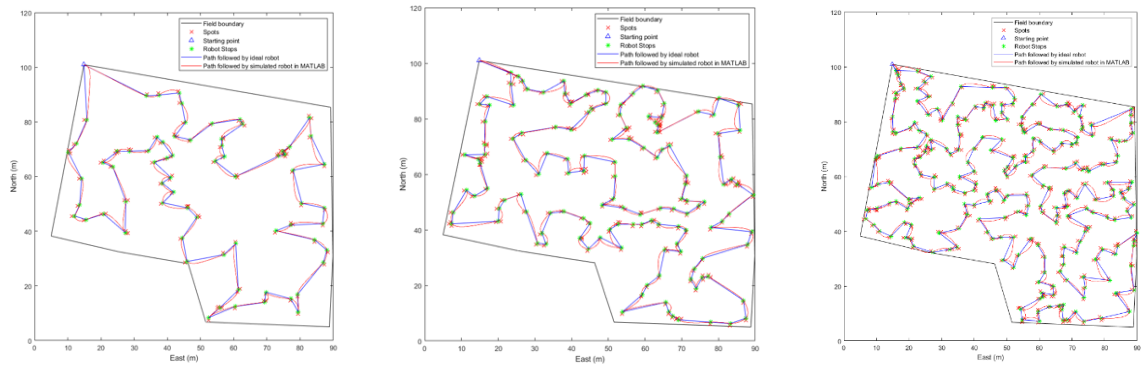


Figure 23 Path of the simulated robot using STPAS method for spot spraying on 100, 200 and 400 spots respectively.

The tables (**Error! Reference source not found.** and **Error! Reference source not found.**) provided summarize the performance outcomes of the proposed STPAS algorithm in a simulated environment for precision spot spraying tasks. The results are differentiated between the idealized calculations of performance (STPAS (ideal)), and the performance achieved using the dynamic window approach algorithm (STPAS (simulated)). The dynamic window approach algorithm is commonly used in real robots for autonomous navigation towards provided waypoints. The benchmark represents the performance of a conventional boom sprayer.

The table indicates that while the distance error remains relatively low, the time efficiency of the simulated robot significantly deviates from the ideal robot, especially as the number of spots increases. This indicates that the actual algorithm, when encountering a higher number of waypoints, faces more complexity which affects its time efficiency.

These results demonstrate the STPAS algorithm's significant contribution to reducing the distance traveled by the AGV for spot spraying applications. Notably, as the number of spots

increases, the relative reduction in distance traveled compared to the boom sprayer diminishes, which could be due to the increasing complexity of pathfinding with more waypoints. However, it's important to observe that while the simulated robot utilizing STPAS algorithm consistently shows an increase in time taken as the number of spots increases, the ideal scenario does not reflect the same rate of increase.

The increased time taken in the simulated robot utilizing STPAS algorithm compared to the performance of ideal robot across all scenarios suggests that there are practical considerations and constraints in the real-world application that the ideal model does not account for. These include AGV acceleration and deceleration, turning time, and obstacle avoidance that aren't captured in simple distance-time calculations.

2.4 Future Research Direction

- Inclusion of payload (pesticides, herbicides) model in the simulator to simulate and calculate the amount of chemicals saved.
- Fabrication of full robotic sprayer for full operation of the spot spraying and quantifying the performance of the spot sprayer in terms of sprayer deposition and operation cost.

2.5 Conclusion

In conclusion, our research undertook a structured and methodical approach to advancing the application of autonomous ground vehicles (AGVs) in precision agriculture. Initially, we developed a MATLAB-based Dynamic Window Approach (DWA) simulator, designed to emulate the performance of real-world robotic platform that we intend to develop a precision spot spraying on. Upon rigorous testing and fine-tuning, this simulator demonstrated a high

degree of accuracy, closely mirroring the time and distance metrics recorded by actual AGVs across various waypoint traversal scenarios. With the validated simulation tool in place, we proceeded to assess the efficacy of our STPAS method. The STPAS algorithm's performance was compared within the simulator across multiple spots spraying scenarios, varying in spot density from 100 to 400 spots. The results were compelling, showing a significant reduction in both distances traveled, and time taken when compared to the benchmarks set by traditional boom spraying methods. These findings open avenues for future work, including the application of the STPAS method in live field conditions and its adaptation for other agricultural tasks. Furthermore, the successful simulation-to-real-world transition establishes a framework for developing more complex AGV systems, promoting innovation in sustainable farming technologies. Furthermore, the empirical evidence of the simulator's performance underlines its capability to serve as a test bed for future innovations in AGV technology. As the complexity of agricultural tasks continues to rise, such tools will be indispensable in ensuring that the algorithms driving these robots are both efficient and reliable.

2.6 References

Breuninger, J. M., Welterlen, M. S., Augustin, B. J., Cline, V., & Morris, K. (2015). The Turfgrass Industry. In J. C. Stier, B. P. Horgan, & S. A. Bonos (Eds.), *Turfgrass: Biology, Use, and Management* (pp. 37–103). American Society of Agronomy, Crop Science Society of America, Soil Science Society of America. https://doi.org/10.2134/agronmonogr56.c2

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, *4*(1), 23–33. https://doi.org/10.1109/100.580977

Jin, X., Bagavathiannan, M., Maity, A., Chen, Y., & Yu, J. (2022). Deep learning for detecting herbicide weed control spectrum in turfgrass. *Plant Methods*, *18*(1), 94. https://doi.org/10.1186/s13007-022-00929-4

Shi, J., Bai, Y., Diao, Z., Zhou, J., Yao, X., & Zhang, B. (2023). Row Detection BASED Navigation and Guidance for Agricultural Robots and Autonomous Vehicles in Row-Crop Fields: Methods and Applications. *Agronomy*, *13*(7), 1780. https://doi.org/10.3390/agronomy13071780

Yu, J., Sharpe, S. M., Schumann, A. W., & Boyd, N. S. (2019). Deep learning for image-based weed detection in turfgrass. *European Journal of Agronomy*, *104*, 78–84. https://doi.org/10.1016/j.eja.2019.01.004