Seed Localisation for Quality Assurance of Seeding Patterns based on Image Analysis and Deep Learning

Okke D. Visser
December 2024
University of Twente - CAES

Abstract—**This project focuses on developing a prototype automated Quality Assurance (QA) system for analysing seed placement in gutters, using image processing and deep learning techniques to replace the current unquantifiable manual inspection. This type of practical application has no comparable research anywhere, and this project researched a unique solution for this problem combining multiple technologies. The system aims to provide real-time analysis on metrics like density, detect seed types on noisy backgrounds, and generate a comprehensive representative image for evaluation. A key challenge addressed in the project was the accurate detection of multiple seed types in varying substrates, focusing on optimising the YOLO-based object detection model for performance and precision. The system leverages image stitching to combine multiple images into a complete image representing the gutters while also integrating duplicate seed detection to ensure no duplicates on overlapping areas of the image. The prototype's ability to generalise with new substrates was evaluated, showing promising results with minimal retraining required. The prototype successfully met the requirements, providing valuable data for seed analysis and showing potential for integration into the production environment. The results suggest that the system can be further enhanced with improved validation metrics, expanded seed datasets, and potentially more powerful hardware to support faster conveyor speeds. Future work will likely focus on refining the integration, enhancing model accuracy, and expanding the system's ability to handle diverse substrates and seed types.**

Index Terms—**Vertical Farming, Seed Localisation, Object Detection, Neural Networks, Deep Learning, YOLO, Image Stitching, Practical Application**

I. Introduction

Seed density is an important factor in farming because it directly influences crop yield and quality [1]. A high density in seed distribution results in competition between plants for nutrients and lower yields, while a low seed density also results in lower yields by having fewer plants growing in the same area. This problem also applies to vertical farms, where the goal is to more efficiently grow and harvest crops than traditional farming methods by using horizontal and vertically stacked layers in environmentally controlled conditions. To further improve the efficiency of vertical farms, seed density should be analyzed and validated to improve crop yield.

The vertical farming company Growy, based in Amsterdam, is currently trying to complete its automated system from planting to packaging. Currently, the inspection and verification of the seed density in the gutters from the seeding machine are still done manually. Manual quality inspection for density verification is often slow and subjective and does not produce measurable metrics for future analysis like seeds/mm^2, but only a pass or fail score.

The popularity of using image processing, computer vision, and/or neural networks for these types of tasks has inspired Growy to try taking the same approach to this problem and design a prototype for automated validation.

The to be designed quality assurance (QA) station has to analyse the seeded gutters from the seeding machine in real-time. It should be integrated into the seeding machine and output relevant data points for real-time validation, logging and reporting for later analysis. A prototype, together with documentation, will also have to be realised to prove the design.

The prototype will capture the seeded gutters from the seeding machine using a controlled camera and lighting setup integrated into the seeding machine. The captured images will represent a section of the fully seeded gutter and be processed in real-time in between captures. The images will be analysed using a deep learning model based on the YOLO (You Only Look Once)[2] architecture to produce all relative seed positions in the images. The YOLO architecture is an actively developed object detection deep learning model that balances accuracy and efficiency to enable the model in real-time mobile applications. The individual images and detected seed positions are combined during runtime to compose the complete image of the seeded gutter and detected seed positions in it. When the gutter has fully passed through the QA station, the final data points will be calculated for validation and logged for reporting and analysis.

While this project does not aim to develop or improve the technologies of image processing or deep neural networks, it does combine these technologies for a specific underexplored application used in vertical farming. By using these technologies to address the challenges of seed density verification in vertical farming environments, this project contributes to filling a gap in practical applications, particularly in scenarios where both speed and accuracy are critical on embedded systems.

This report will first discuss the deeper background behind the project to showcase the problem and limitations. Then, it will analyse related work to research what other solutions have been used for similar problems. With the gathered information, the chosen design of the prototype QA station will be explained. Then, the experiments with their results will be discussed, and a conclusion will be drawn on how the project went, performed, and what could be improved.

II. Background

Growy is currently in a transition phase where after prototyping the most important and experimental parts of the complete system, it is ready to implement all gathered knowledge in a better scalable design at a new farm location. The improved system at the new farm, located at CTPark Amsterdam City, still needs human operators for some semi-automated processes like packaging and general supervision, but currently, the system needs a human operator for validating each of the seeding machines used in the system. This process is not efficient and reliable, especially when the farm will be scaled up. Using image processing in combination with deep learning models to replace human operators should improve reliability, scalability, and provide additional metrics for data analysis.

A. *Seeding machine*

The quality assurance (QA) station is part of the seeding machine, seen in figure 1. This machine is responsible for taking the gutters, 2.5m long and 10cm wide metal trays with ID tags, and then laying a thin substrate on the bottom, placing the specified seeds on the substrate with different densities based on the seed type and watering the gutter as well. These seeded

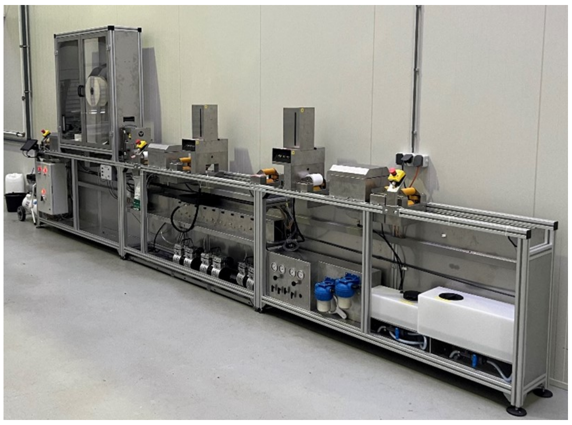

Fig. 1: Prototype of seeding machine used at old Keienbergweg office

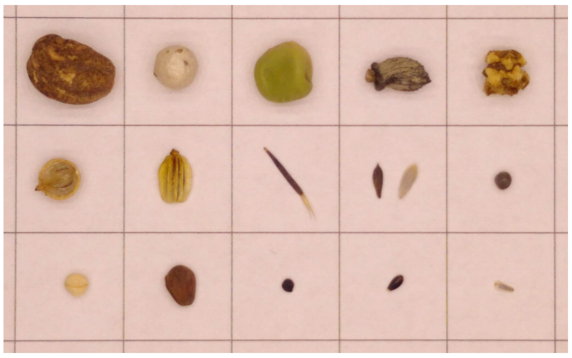

Fig. 2: A photo of 15 different seed species that have large visual differences on 1cm grid paper using a controlled lighting setup

gutters can then be moved into the growing cells where they are stacked in multiple vertical layers for space efficiency. The QA station sits on the end of the seeding machine and before the conveyor that takes the gutters to the growing cells. This way the quality station can reject gutters that do not fulfil the criteria and log data for every gutter that passes through on certain metrics.

Current specifications of the QA station have the maximum speed of the gutters moving through it at 50mm/s with 1-2s deciding time where the gutter is paused. The QA station has to operate in real-time because the decision of pass or fail needs to be almost immediately made after the gutter has completely gone through the QA station to direct the gutter to the growing cells or to discard it to not slow down throughput.

The integration of the QA station with the rest of the seeding line is composed of two parts. The first part is the digital information of what seed type is being used with additional information like conveyor speed. This transfer of information will be done with a CAN bus using CANopen as the communication protocol[3], with the QA station being a slave node.

The second IO for the QA station will be 24V signals. One input announces a gutter coming to the QA station, and separate output signals for pass or fail conditions to the conveyors at the end of the QA station.

The seeding machine is capable of planting different seeds and substrates. For the seeds, Growy uses at least 50 different species, with a lot of visual differences between some of them. Most of the seeds resemble a brown ball with diameters of 1-5mm, but some have different colours or shapes that do not resemble others. Figure 2 shows 15 of the more extreme seed differences. The smallest currently known seed is 0.8mm in diameter. Growy will continue to introduce new seed species in the future production line and discontinue others.

Different types of substrate can also be used by the seeding machine. Growy used a biostrate substrate in the past for most of the plants, and it had a consistent white texture but was switched

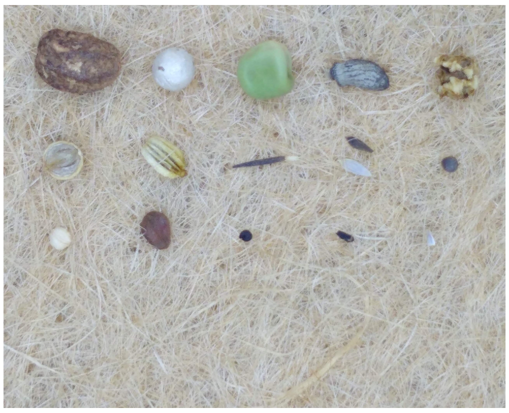

Fig. 3: Jute substrate texture with the same arrangement of 15 seed species placed as figure 2 with a controlled lighting setup. The bottom-right seed especially blends in very well with the substrate texture.

due to supply issues to a light brown jute substrate that has a noisy texture with loose fibres. This new substrate will make the seed detection harder to perform due to the sometimes similar colour as the seeds and rough texture of the substrate as shown in figure 3. Growy is always looking for better substrates, so the substrate could likely change again due to new supply issues or because different substrates are used per seed species to improve yield. This should be kept in mind to potentially future-proof the realised system.

B. *Output Metrics*

From discussions with the data team at Growy, a few data points have been agreed upon to be useful for the validation at the QA station and later data analysis that can be provided by the QA station. These metrics/data points are:

- Top-down image showing the complete 2.5m seeded gutter for later (visual) inspection/referencing
- Used configuration and circumstances of QA station (Image acquisition parameters, seed species, conveyor speed, timestamp, etc.)
- Basic parameters of seeds (Size, species, other)
- Total number of seeds in the gutter
- Relative coordinates of all seeds in the gutter
- Density map of gutter to determine a good distribution. (Places with not enough seeds, clumps or more dense spots, etc.) The minimum and maximum density spots could be used in validation.
- Visualization of density map in the form of a heatmap

Storing a complete image of the gutters, while not providing any metrics or data points, is a helpful tool to provide confidence in the performance of the QA station and give the possibility to inspect the gutters at a later date for data/metrics not gathered at that time. The used configuration and circumstances together with seed parameters are also very important for data analysis with possible connections from this data, like conveyor speed and seed size, to crop yield.

The relative coordinates of all the seeds in the gutter are the main data from the QA station that can determine the quality of the sown gutters by counting the total amount of seeds and calculating local densities. This data from the seed positions can also again be used to link crop yield to the output of the seeding machine in later analysis.

III. Related Work

The main objective of the project is a form of object localisation, often also referred to as object detection. From literary research, there are two main popular methods.[4]

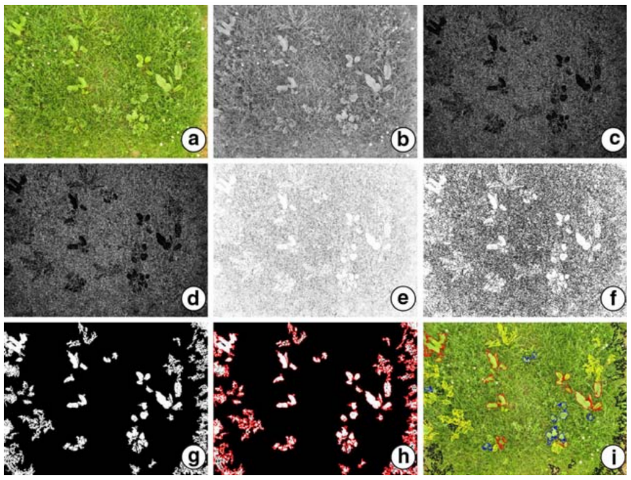

Fig. 4: An example of traditional logical methods in one study that apply grayscale (b), gradients (c), binarization (f), and morphological opening (g) in sequential steps to a RGB image [10]

The first is the traditional object detection methods that use multiple standard mathematical or algorithm-based steps together to extract the features of shapes like circles [5]. Often used preprocessing steps in these methods are; changing colour space to grayscale, blurring, calculating gradients, binarization and morphological functions [4][6][7].

With the image preprocessed, methods like Canny edge detection, circle fitting, contour finding or segmentation can be used to find objects and their parameters, like size and position[8][9].

This approach of using logic and mathematical-based algorithms often results in very fast processing for real-time applications with also enough accuracy to be an attractive solution [11]. Especially for circular shapes, there is a lot of literature with slightly differing approaches using these steps [5][12]. These approaches do also not care about dataset size and perform the same on small datasets as on large datasets.

Most of these steps often rely on good differentiation in one or multiple colour channels between the objects and the background of the image. Most methods therefore use a solid colour for the background or have a backlight when capturing the images[13][14][15]. Most of these methods also are often tuned for specific shapes or objects and require manual tuning of the parameter for most of the methods and filters used to best fit the to-be-detected shape or object. an example of object detection using mathematical and algorithm-based steps can be found in Figure 4.

The second popular object detection method is based on deep learning with CNNs (Convolutional Neural Networks).

There exist many different forms of deep learning-based object detection, some examples are Faster R-CNN, YOLO (You Only Look Once), SSD (Single Shot Detector), and RetinaNet [16][17][18][19]. While we are only interested in the bounding boxes of the objects in this project and not the classification, these same object detection CNNs are used for this purpose in the industry.

The CNNs, or models, use the raw image data, or a grayscale version, instead of features that are extracted from the image by other algorithms. This makes the models more flexible with detecting different shapes and objects since the models can fine-tune their feature extraction instead of relying on handcrafted feature extraction that takes time to fine-tune and is often less effective [20]. This therefore generally makes the models better able to handle backgrounds and changing environments than the traditional methods, provided that the model has a big enough dataset to train on.

Deep learning models are very complex architectures that can require more powerful hardware for training and inference compared to traditional methods. Using these on resource-constrained devices can be a challenge. The mentioned models are however the most efficient for object detection compared to other deep learning methods as they were designed for real-time applications on mobile devices. [21]

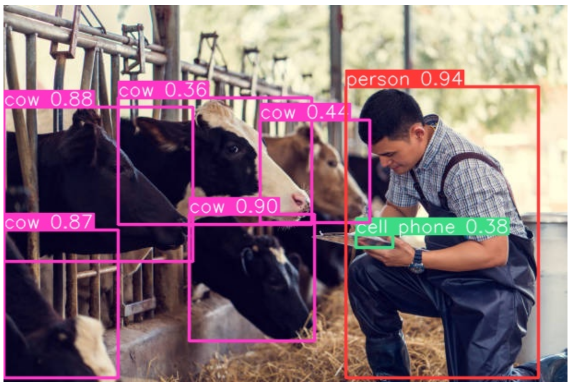

Fig. 5: Example of YOLO-based object detection output from an image of a farmer with cows. The output shows the bounding boxes of the detected objects, the classification of these objects, and the confidence for these classifications.

The biggest downside to using deep learning models compared to traditional methods is the required training dataset sizes. Achieving high performance with these models often requires a large amount of manually labelled training data that might not be readily available.[22]

A helpful tool with deep learning models to combat this issue of required training data is transfer learning, where the models are pre-trained on large general datasets and later re-used for specific tasks that then require less training data and have better generalisation compared to when trained on task-specific data. These pre-trained models are widely available for most deep-learning models.

From research analysis, it seems that using YOLO (You Only Look Once) as the base model for deep learning-based object detection is the best option for this project as it has the best trade-off for efficiency, performance, and accuracy [23][24][25][26]. An example of output provided by YOLO is seen in figure 5.

There is a lot of literature about using algorithms to detect positions and/or classify seed species from image data, but they all differ in important ways from the situation of this project. The studies either try to classify the complete image with no object localisation [27], or more commonly, the studies make ideal situations for the image acquisition by having a controlled lighting setup and using a solid colour as the background like white or black [28] or use a backlight [15]. This often results in the studies being able to use traditional methods like binarization for object localisation to extract bounding boxes of the seeds and use a deep learning model for the seed species classification of the cropped images [29][30][31].

In our application, the seeds cannot be always isolated in the same way, as the background is a noisy substrate with sometimes very similar colours and wildly different shapes between seed species. So while the subject of the studies is the same or similar, the situation differs greatly from lab conditions to a production environment. Another difference is that in this project, the focus is on object localisation and not classification, as the seeding machine already knows what species is being used. So while the research part of the research in these papers can be used, they can't be used as the complete solution for this project.

As mentioned before, most studies use a controlled lighting setup where the light comes from single or multiple predetermined angles with a controllable brightness[32][33]. Most of the setups also remove any ambient lighting to ensure that the outside environment has no influence on the image acquisition[24]. This is done as differences in lighting angles or brightness can have a very adverse effect on the detection or classification after image acquisition[34][35]. While some deep learning methods can learn to overcome this issue, it is still preferred to remove this variable and digitally alter the images before training as a form of data augmentation to generalise the model if needed [36].

For this project, the to-be-analysed objects are the 2.5m long gutters. It is unreasonable to assume that the complete image of these gutters will be gathered in one photo or frame, and instead, the complete image can be composed of multiple smaller images and stitched together at the end of the seed localisation. This stitching method is already widely used for image processing[37][38] and has also seen some implementation in deep learning models like with SAHI (Slicing Aided Hyper Inference)[39]. With a rough estimate of the position difference between images from the conveyor speed and minimal expected projective and affine transformations[9], this should make the image stitching quite fast and able to be performed in real-time.

IV. Design of the QA Station

The design of the QA station is split into multiple components, including software and hardware. To understand the overall design of the QA station, Figure 6 shows how the overall design works and how these main components work together.

Discussions with Growy have vaguely defined the requirements of the prototype QA station, with the intention that it can prove the technology and design and that later iterations can use this prototype to better inform them about specific targets for the requirements.

The requirements for the QA station are as follows:

- Design for a prototype QA station and its realisation and documentation.
- Analyse the seeded gutters from the seeding machine in real time. (Max. 2s execution time between images)
- Integrated into the seeding machine. (The seeding machine can provide data about current production, and QA can give pass/fail signals to external machines)
- Works with multiple seed and substrate combinations
- Provide relevant data points
  - Complete top-down image of 2.5m seeded gutter for later (visual) inspection/referencing. (The image represents the actual used gutter well with no distortions or artefacts as if one image was taken with an isometric camera)
  - Used configuration and circumstances of QA station. (All metadata of the lighting, camera, and seeding machine settings)
  - Total number of seeds in the gutter. (Both recall and precision for the detected seeds should be equal or greater than 0.95)
  - Relative coordinates of all seeds in the gutter. (Position relative to the gutter and size for every seed detected)
  - Density map of gutter to determine a good distribution. (Minimum and maximum local density metrics)
  - Visualization of density map in the form of a colourmap.
- Use the data points after the gutter has passed through the QA station to validate the seeded gutter
- Log all gathered data for later reporting and analysis

Every component will be discussed in its subsection and will include the design choices for that component.

A. Hardware setup

Figure 7 shows a photo of the testing setup, with the subcomponents processing unit, lighting setup and camera module further explained.

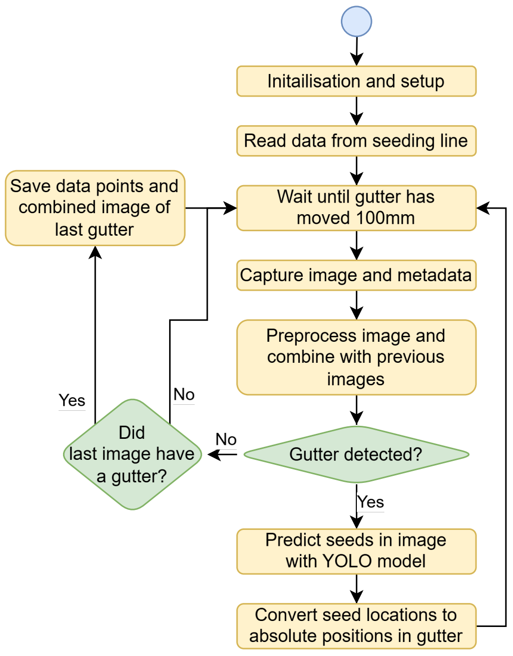
Initailisation and setup
Read data from seeding line
Wait until gutter has moved 100mm
Capture image and metadata
Preprocess image and combine with previous images
Gutter detected?
Yes
Predict seeds in image with YOLO model
Convert seed locations to absolute positions in gutter
No
Did last image have a gutter?
Yes
No
Save data points and combined image of last gutter

Fig. 6: Overview of the design flow of the QA station prototype

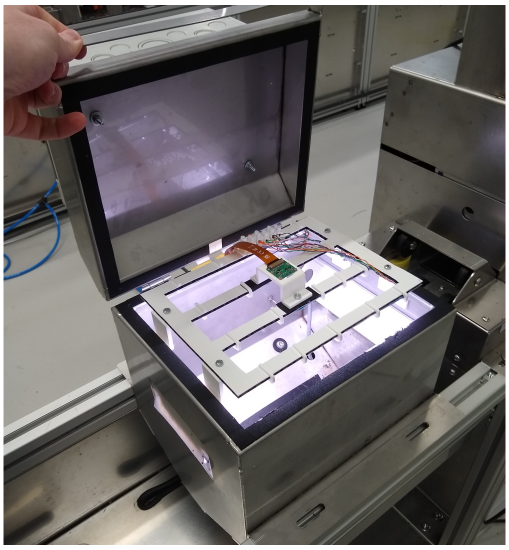

Fig. 7: Hardware setup of the prototype

1) Processing unit: For the initial prototype of the system, a Raspberry Pi 5 (8GB RAM) has been chosen as the main processing unit, as it is expected to have enough performance to run the YOLO models in real time[40] with additional software. This also comes with the benefits of being cost-effective, easy to use, and lots of online documentation. Training of the YOLO model will likely take place on another machine with more performance and dedicated hardware like a PC with an NVIDIA GTX 1070, or one of the servers from the University of Twente. The Raspberry Pi will be supplied power by a PoE HAT since Growy has spares and their infrastructure is built for PoE.

2) Controlled lighting setup: From some initial tests and literature research, the lighting environment can majorly impact the performance of certain methods like object detection or similar image processing methods. To avoid this problem and to try and improve performance, a basic controlled lighting setup will be built as part of the QA station. This will give the possibility to control brightness and colour during testing to try and find the best parameters for image quality overall, and possibly per seed species if time permits.

The lighting setup will have to be built in a 'closed' box to

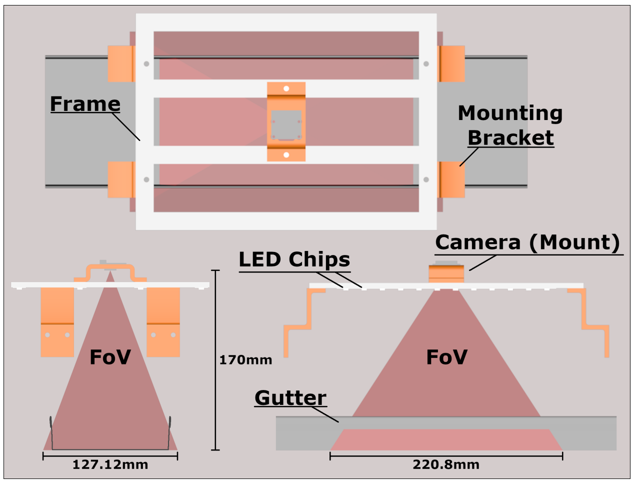

Fig. 8: Schematic of lighting and camera hardware

remove lighting from the outside environment. The seeding machine already has multiple of these closed boxes on its production line for watering the gutters to contain the sprayed water. These boxes can be easily opened at the top for maintenance. One of these old boxes will be repurposed for the needed housing of the camera and lighting setup.

To provide the lighting inside the QA station, regular 24V RGBW LED strips will be used. The seeding machine already provides 24V to every module, so using this voltage will be power efficient and simple with no need for a power converter. The reason for RGBW LED strips is so that the RGB LEDs can be used for testing with coloured lighting and to give better colour contrast for some combinations of seed type and substrate, and the white LEDs for simple white lighting with greater efficiency and brightness. The controlled lighting of the QA station is implemented by using four 200mm strips of 24V RGBW LEDs with 12 LED chips each. The Strips are placed with 51mm space in between so that the chips are spread over a 165x183mm area. This is done so that every position of the 100x100mm area captured by the camera has a light source coming from every angle and should therefore have no obvious shadows or dark areas caused by using a single light source for example. The schematic for the LED placement can be found in Figure 8

The brightness of the LEDs is controlled by MOSFETs driven by PWM signals as this was cheaper and easier to do than finding and buying a commercial 24v RGBW LED controller with a communication interface for a Raspberry Pi. Originally, the Pi was meant to directly switch the MOSFETs with its PWM outputs, but it was later discovered that the Raspberry Pi 5 only has 2 separate PWM channels, and therefore, an Arduino nano is used as a PWM extension board. The PI sends 4 bytes, one byte per colour channel brightness, to the Arduino using I2C, and the Arduino then uses the received values for the PWM duty cycles. The 24V supply for the lighting in the prototype is a spare DIN-rail PSU that Growy had left over. Later production versions can use the 24V used by the seeding machine.

The addition of UV (blacklight) or IR lighting for testing could also be implemented, but some initial experiments concluded that UV light does not show any distinguishing information like fluorescence between the substrate and tested seeds, and the expectations for IR lighting are the same. So it is a possible area for further research with for example a multispectral camera.

3) Camera module: To successfully capture the features of the seeds, 10 pixels per mm (100px/mm^2) is assumed to be enough detail since the smallest currently known seeds are 0.8mm in diameter. If a section of the gutter is captured with the full width visible, which is on average 105mm, this would result in a minimal image size of 1050px. For the camera module used in the prototype QA station, a Raspberry Pi Camera Module V3 has been chosen. This camera supports 4608 x 2592px resolution photos, 1080p50, 720p100, and 480p120 video output, and handy features like focus control [41]. The reason this camera module has been chosen for the prototype is the fact that it is widely used in the industry, including at Growy, because it is relatively cheap, and also has lots of support documentation online. The camera, most importantly, can also capture images and video with high enough resolution for our assumptions, with resolution to spare in case the assumptions were on the low side. The Camera Module V3 is also able to dynamically change its configuration during runtime with commands from the Raspberry Pi to, for example, adjust white balancing or focus distance. After the prototype has been made, later versions can choose a more specific camera to cut costs or improve performance if needed.

The Camera Module V3 needs to cover the complete width of the gutter and to leave some additional headroom for imperfections with the placement of the camera module, like being slightly rotated, the camera is placed about 140mm above the centre of the gutter looking directly down. This results in the smallest coverage dimension of 125mm with the 41degree lens of the camera V3, with a maximum of 25mm of it overlapping between images. The placement of the camera module can also be seen in Figure 8.

Once the controlled lighting and camera setup is built, some calibration experiments will be performed to test out different parameters for the lighting and camera settings like ISO, white balancing, and focus to find the best image quality with the setup.

B. Image acquisition

The camera captures images at a calculated interval derived from the conveyor speed of the seeding line so that it captures a frame for every 100mm that the gutter travels through the QA station. For a speed of 50mm/s for example, which is the default speed of the seeding line, this interval would be 2s. The image is retrieved at full resolution so that the later preprocessing steps can make full use of the raw data instead of transforming an already resized image. The metadata is also retrieved from the camera together with the image. This metadata includes some very useful data like the exact timestamp of the capture in nanoseconds or other information for later analysis like the used configuration, white/colour balancing and sensor temperature.

C. Image processing

Before the captured image is used for seed detection, it needs to be prepared. For a deep learning model application, preprocessing is not that extensive and often just comes down to cropping, resizing, and normalising the images. This is also the case for this project, as YOLO does not need a lot of preprocessing and is often pre-trained on the RGB colour space.

Besides the preparation for the YOLO network, the separate images of a single gutter that are gathered by the camera setup need to be combined to make a fully complete image, as determined by the discussed requirements of the QA station. To accomplish this, the images will need to be stitched together between taken images or during the pause between gutters. The relative positions of every frame compared to the complete gutter are needed to convert the positions of the detected seeds to their absolute positions in the gutter. Additionally, the beginning and end of the gutter will be detected in the first and last captured images of the gutter, respectively, to crop out the unnecessary pixels that just show the background of the QA station and to always set the horizontal origin of the seed coordinates on the gutter edge. This gutter detection can also be used to detect if there is a gutter present in the captured image at all and to discard it if not.

1) Undistortion and transformation: The image preprocessing consists of two steps. The first is removing the camera lens distortion from the images. This lens distortion is minimal but can be seen at the edges of images when overlapped together for the total image of the gutter. The second step is applying a transformation matrix to correct any perspective warping or rotation introduced by the imperfect camera placement. This

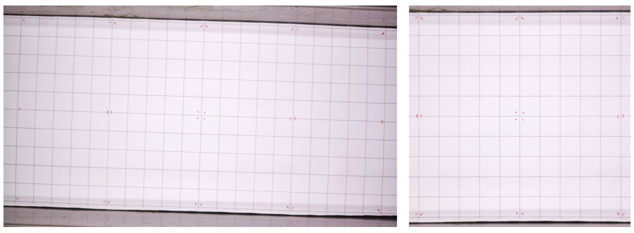
(a) Original calibration image (b) Calibrated image

Fig. 9: Comparison between original captured and transformed images used by YOLO model. (Not to scale)

transformation matrix also scales the image to the final resolution used by the YOLO model, which was made dynamic in the preprocessing. A grid with 10mm spacing is placed in a gutter under the camera at the same distance as seeds on a substrate, and some images are captured and saved. These images are then loaded into MATLAB, and by specifying the 4 corner points of a 100x100mm square in the calibration images and using the `undistortImage` and `estgeotform2d` functions, the camera intrinsics and transformation matrix were extracted. The images cover an approximated 110x110mm area after the transformation to leave some overlap between images of the gutter. The resulting calibration is shown in Figure 9.

2) Gutter detection: An additional task the image processing performs as mentioned before is checking the transformed images for any activity, meaning if there is a gutter present in the image. Without activity, the software pipeline can discard the image and wait until the next capture interval. If a gutter is detected, the horizontal point where the gutter starts can be used as part of the origin for the seed locations, and the start of the combined image of the gutter. Once the whole gutter has passed through the QA station, no activity will once again be detected and this signifies the end of the gutter to the pipeline. Again, the point where the gutter ends is saved to end the combined image. This detection is achieved by two physical black stripes in the background of the camera inside the enclosure. Once the gutter passes over these black bars, the software detects a change in brightness in these parts of the image. Checking where this change in brightness begins or ends results in the horizontal points.

3) Image stitching: With the transformed images, a combined image showing the whole gutter can be made and saved for later reporting and analysis. To construct the combined image, the latest transformed image is appended to the combined previous images of the same gutter during runtime. Since the only difference between the images after transformation is the horizontal position, there is no need for algorithms like SIFT. The horizontal difference can be approximated by calculating the time interval between images using the timestamp metadata and the conveyor speed.

With this approximation, the final horizontal movement can be fine-tuned by offsetting the approximation and calculating the difference of the overlapped areas for multiple offsets. The offset with the smallest difference is then used for the final horizontal movement. The latest image is then appended onto the gutter image using horizontal movement and linear fading on the overlapping area. The fine-tuning is needed because it was observed during the project that the speed of the gutter is not perfectly constant. The horizontal movement between images is also saved in the output metadata for that image for later reference and used for calculating the absolute position of predicted seed positions later in the software pipeline. Pseudocode for the image stitching can be found in Appendix B (Algorithm 1).

D. Seed localisation

The YOLO deep learning model has been chosen for the seed localisation method. This decision is mainly based on generalisation and flexibility.

Unfortunately, the main problem for localising the seeds on the substrate is the substrate itself. Because the QA station cannot change the seed positions, it cannot easily isolate the seeds from the substrate in any images it captures by using a solid colour background. This problem is also aggravated by the many different seed species that have different colours and shapes. Traditional methods would find it hard to successfully detect the different species of seeds on the substrate without changing multiple steps in the method between seed species. In contrast, a deep learning model can learn to generalise the problem with enough training data. With the throughput of a single seeding machine in operation aiming to process 400+ gutters per day, resulting in a rough estimate of 10,000 images with a total of 2,000,000 seed instances per day, training data should not be a problem. The initial labelling of the data will be very time-consuming, but eventually, the labelling could transition to inference-assisted labelling and only modify wrong predictions.

The second main reason is the flexibility. For traditional methods, most filters and algorithms have parameters that would have to be manually configured, like kernel sizes, thresholds, etc... In the case that a new seed species is added to the production line, the QA station would need a lot of manual configuration to support the new seed species. The YOLO algorithm would, however, only need a few gutters of training data to improve the detection accuracy of the new species by utilising the already learned features of previous species as the basic foundation. The hope is that the model generalises so much that new seed species will automatically be detected without the need for any new training data, but this seems unlikely.

For this project, the YOLOv8 models by Ultralitics will be used [2]. They have different size models (3.2M to 68.2M parameters) pre-trained on the COCO dataset [42] with 80 trained classes.

Multiple seeded gutters of different seed species will be manually labelled. The dataset will be split into a 7:2:1 ratio between training, validation and test data. With this dataset, the model will be specifically trained for seed detection.

The detection and localisation of the seeds is implemented in two main steps. First, the YOLO model predicts seed locations relative to the given image, and the next step converts the relative locations to the corresponding absolute position in the gutter while removing duplicates due to overlap between images.

1) YOLO configuration: YOLOv8 provides 5 different pre-trained models (nano, small, medium, large, extra large) with parameters counting from 3.2M to 68.2M, and every model has the option of using any square input resolution while it is a multiple of 32. Ultralitics also support exporting to many formats of YOLOv8 like PyTorch, TorchScript, ONNX, NCNN, and more. For this project, a Raspberry Pi 5 is used, so the format NCNN was chosen as this is a format highly optimised for embedded platforms and recommended by Ultralytics for the Raspberry Pi[40]. The use of the NCNN format was also validated through testing.

The main choices for the YOLO configuration were what model size and image resolution to use within the limits of the project while giving the best results. Choosing what model size and resolution to use for the inferences during runtime is very important, as this directly impacts accuracy and inference time. A test to compare the different models and image sizes which can be found in section V.

The choice made from this test was to use model size nano with 1280px as the image size. The results and reasoning for this decision can be found in section VI and section VII. From the validation metrics of the chosen configuration, the best confidence and IoU (Intersect over Union) thresholds can be found and then later used for inference in the actual software pipeline.

Another choice made for the YOLO implementation was the fusing of seed classes. This means that instead of labelling all seed types as different classes, the seeds are all labelled as a single class, and therefore, classification is not performed. The result is that the YOLO model will only perform object detection. At the beginning of the project, the assumption was made that fusing the seed classes would improve the generalisation of the trained YOLO models. Later on, this decision was validated through tests and experiments, as shown in section VI.

In the software pipeline, the model is loaded during initialisation, and one inference is immediately performed to ensure the model is fully loaded and later inference times are as expected. When a prediction is made for a given transformed image, the relative bounding boxes are retrieved and passed on to be localised in the gutter.

2) Relative positions to absolute coordinates: The relative seed positions from the YOLO predictions are transformed using the horizontal positions gathered from the image stitching to the absolute coordinates in the gutter. Because there is some overlap between subsequent images, some seeds may be detected twice by the YOLO predictions. These duplicates need to be removed to not influence the seed density metrics used to validate the gutters. This is accomplished by checking the IoU (Intersect over Union)[43] for all seeds in the overlapping areas, and if this exceeds a threshold, the average position for the two seeds is used and the duplicate is removed. This software to detect and remove duplicate seeds has been tested, and the results can be found in section VI. The pseudocode for this software can be found in Appendix B (Algorithm 2).

Once the gutter has fully passed through the QA station, one last check is performed to ensure that all seed positions are within the gutter. Some false positives can occur on images captured when the gutter is only partly visible and detected in the background.

E. Data logging and reporting

The gathered data from the seed localisation will need to be stored and used to calculate the other relevant metrics for the validation. The metrics that need to be calculated from the seed positions are a seed count, average density, density map, and the maximum and minimum of this density map.

Once a gutter has fully passed through the QA station, the relevant data metrics are calculated and compiled together with all gathered metadata like the camera configuration. Simple metrics like seed count and average density are easily calculated. For the density map and derived minimum and maximum densities, convolution is used on a black image with white pixels representing the seeds. A custom kernel is made with a distance function based on a given radius. The kernel values are divided by its 'volume' within the image to compensate for different radii and image boundaries. From this density map, the minimum and maximum are gathered and a colormap is made for easy visualisation. An example of a generated colour map can be seen in Figure 10. All the calculated metrics can then be used for validation using a target value and an acceptable deviation. The metadata with metrics, the total image of the gutter, and optionally the individual captured images for later training are then uploaded to cloud storage for later reporting or analysis.

F. Integration

While the QA station mostly operates independently, some integration with the seeding line is necessary. Eventually, the QA station needs to communicate with the seeding line to request both the conveyor speed to determine the image capture interval and the sown seed type to potentially change the lighting and camera configurations to optimise the captured images. Growy communicates with the different modules of the seeding line with CANopen, a communication protocol designed for embedded systems used in automation based on CAN. During the project, there was no time to integrate the QA station with the CANopen communication of the seeding line, but the preparations have been made by using a virtual CAN channel. This way, the integration requirement of reading from the seeding machine can theoretically be fulfilled. The QA station hosts its own CANopen[3] node using Lely CANopen[44], with registers for the conveyor speed and seed type on a virtual CAN channel so that the software pipeline can access these values during runtime with the *CANopen for Python* library[45]. The master node of the seeding line can then be emulated by manually writing the values to the registers. Once it's time to integrate the QA station with the seeding line, the virtual CAN channel can then be replaced with a physical CAN channel of a CAN HAT for the Raspberry Pi 5 and Growy would need to add the integration to the master node of the seeding machine for providing the specified values to the slave node registers.

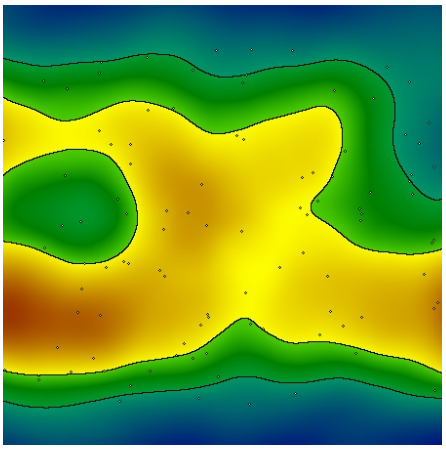
Fig. 10: Example of a partial colour map to visualise the seed density of the gutter. The markers are individual seeds, with black lines dividing the acceptable regions (green) from the rest. A full example can be seen in Appendix A

Using the seed type information from the (emulated) seeding line, the software will look up the corresponding configuration for the camera and lighting setup, as well as the target metrics for validation in a JSON config file. This configuration information can then be used to capture the best data from the seeded gutter and to pass or fail the gutter based on the calculated metrics.

The QA station will also need to signal the conveyor after the QA station or the seeding machine itself whether the seeded gutter passes the QA tests. The current prototype of the QA station is not capable of sending signals to the seeding machine and, therefore, fails the initial requirement for using the gathered data for validation. Because time was limited and other components of the QA station required more time and effort, the relatively simple hardware to output 24V signals based on the validation metrics was not included in the prototype. Growy also has no current use for the validation output signals as the QA station prototype was not intended to immediately replace the current human operators.

V. Tests and experiments

This section explains the testing setup for the validation tests in this report. Every test will explain why the test was performed and how. The results of the tests can be found in section VI.

A. Image processing

To test and validate the image processing of the project, a test will take two images captured in a sequence of the same gutter and combine them after the pre-processing steps using the implemented image stitching. The conveyor speed during the image acquisition is also given in this test for the approximation of the horizontal movement. This will show if the software can fulfil the requirement of generating a complete representative image of the gutters. The images include the timestamp metadata of the captured images and can be seen in Figure 11.

The test results will show the approximated offset between features in both images and how well they overlap. The same will be done but with the fine-tuned offset. Afterwards, the two images will be stitched together and shown as the resulting image. The test will be successful if there are little to no imperfections in the image stitching or the overlapping features using the fine-tuned offset.

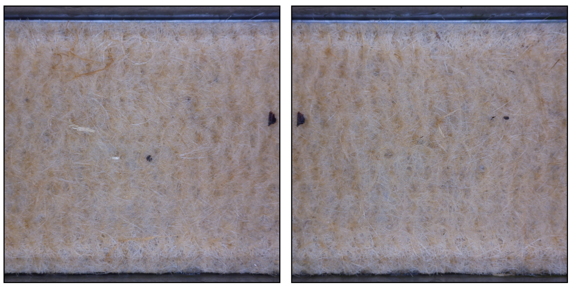

Image Nr. 9 **Capture time: 12,987,224µs** — **Image Nr. 10** **Capture time: 14,728,976µs**

Fig. 11: Two images captured of the same gutter with the sequence numbers of the images and the capture time in microseconds

B. *YOLO model configuration*

This test aims to find the best combination of YOLOv8 model size and image resolution for this project. The main metrics used to find the 'best' configuration were the inference time and detection accuracy (mAP50 score) of different seeds. This test will also determine if the QA station's real-time requirement is feasible.

Every model size is trained with 3 different image sizes, 640, 960 and 1280px, to find the best accuracy within the inference time limitation. 640px is the default size for YOLOv8, and 1280px is double this while also approximating the beforementioned assumption of 10px/mm being good enough. 960px was also included to better understand the scaling and impact of accuracy and inference time with different image sizes. Once the models were trained, they were transferred to the Raspberry Pi 5 and exported to the NCNN format to better represent the final model used.

The maximum theoretical limit of the inference time is 2s with the standard conveyor speed being 50mm/s and the images representing 100mm of the gutter, but to accommodate the runtime of the other software and leave some headroom, an inference time of 1s will be chosen as the upper limit. Therefore, any configuration that exceeds this limit will not be considered. The first inference time will be excluded, as the model is fully loaded here, which increases inference time, and this will also be done in the prototype software. While the model size has little to no impact on the performance of other software components, image size does have a relevant impact, especially in image preprocessing. This impact will also be considered when deciding what configuration to use.

For these tests, a dataset was composed of 131 total images with 15,094 labelled seeds and 4 different situations ('Bright and Spicy' seeds on biostrate, 'Amaranth' seeds on biostrate and 'Mirco Thyme' seeds on both biostrate and jute). The *val* function of the YOLO Ultralytics library was used for the accuracy metrics.

The models were trained on the University of Twente's EEMCS-HPC cluster with a maximum of 1000 epochs. For every configuration, a combination of the base model and image size, the average and standard deviation inference time for 100 inferences is saved in ms for a single image of the specified image size on the Raspberry Pi 5 using the NCNN format for the model. The mAP50 scores give the mean Average Precision for the detections made by the trained models using the specified configurations with at least 50% overlap using IoU. The mAP50 is given per seed type used for training to give better insight into how different configurations impact specific types. The average for these mAP50 scores is also given to give an overall indication of the precision of the whole model. The reason this was chosen over the total mAP50 score is that the seed types do not have the same data size in the dataset, so using an average is more representative of the model precision.

C. *YOLO model format*

To validate the choice of using the NCNN format for seed detection and the real-time requirement feasibility, a small test was performed to compare performance and accuracy between the native format PyTorch and the exported NCNN format. Because the models are trained in the PyTorch format, the trained model with the chosen configuration ('nano' base model and 1280px image size) was transferred to the Raspberry Pi 5 and locally exported to the NCNN format. Both models were then tested with the same methods as for the model configuration, running 100 times to gather the inference time metrics and separate accuracy tests for every seed type used in the dataset and combined for the average mAP50 score.

D. *Single or multi-class model*

Another assumption made during this project is that using a single classification for all seed types instead of having separate classes for every different type would help the generalisation of the models. To validate this assumption, a test was performed between two trained models using the chosen model configuration to compare accuracy and inference times between separate classifications between seed types and a single class. The datasets used are the same as the model configuration test, with all situations classified as different classes. One model was trained using the *single cls* option from the Ultralytics library enabled, which fused the classes into one, and another with this option disabled. The models were then again transferred to the Pi and exported to the NCNN format. The same tests as the YOLO model configuration and format were run.

E. *Generalisation Test*

In an ideal situation, the trained model can accurately detect any seed type on any substrate used by Growy. Realistically, this is hard to achieve, but the goal should be that the model is generalised enough that a minimal amount of training data has to be prepared and further trained on to have accurate results. Since, during the project, the seeding line only used a few seeds with different characteristics, an experiment was performed with the new jute substrate to find out how quickly the model could accurately predict seeds on this untrained substrate. Another reason for testing generalisation for a change in substrate is that this would affect all seeds used in production if the accuracy lowers for the new substrate instead of a single new seed type.

For this experiment, a model was trained on the dataset without any images of jute substrate as a baseline, and then the model was trained further from this baseline with different amounts of jute substrate images (1-4) to show how much the data size of a new substrate influences the accuracy of the models. The test results can then inform how much initial training data should be further trained on when a new substrate or seed is used.

F. *Seed duplicate detection*

To validate the software component responsible for the detection and merging of duplicate seeds, a test was performed with two sequential images with a large overlap. For this test, the saved metadata of the predicted seed positions per image was extracted and again run through the software to find any problems. To find potential problems, the seed positions are annotated onto the image with red circles for seeds detected in the first image, and blue crosses for detections in the second image.

Once the software has merged the two lists of seeds, the merged seeds are annotated with black squares on the image, with unmerged seeds that have no corresponding duplicate between images marked the same way as before. Looking at the resulting annotated image can then reveal any potential problems with the detection and merging of duplicate seeds.

G. Accuracy of system

The previous tests validated separate components of the software and prototype, so one system test will also be performed to validate the whole system.

To prepare for the test, the QA station prototype is mounted on the seeding machine and calibrated with the methods described in section IV: Image Processing. The seed type and conveyor speed variables are manually changed by the emulated CANopen seeding machine master node. The software is then run until a fully seeded gutter has passed through the QA station. The resulting uploaded metadata JSON file, total image, and density colourmap of the seeded gutter are then retrieved from the cloud server.

The combined total image is then checked for any abnormalities or artefacts from the image stitching. For the data points used for validation, the given seed detections are first compared to the 'true' detections that have been manually performed. This will give the total amount of missed seeds (false negatives) and false detections (false positives) together with the precision, recall and F1 score for the prototype. The other metrics, like the seed densities, are calculated by hand or using part of the software with the manually reviewed seed position list. This will give error differences to see how well the prototype calculates the validation metrics.

VI. Results

All the test and experiment results from this report will be explained here with some observations. Discussions about the results can be found in section VII.

A. Image processing

The image stitching software tested with two images captured in a sequence of the same gutter and can be seen in Figure 11. The image includes the timestamps of the captured images and results in an interval of 1.741752s. This test used a conveyor speed of 58.82mm/s as the default speed, resulting in an approximated offset of 1188 pixels between images. Using the fine-tuning software, which looks at the difference between the overlapping areas between images, the resulting offsets becomes 1203px. Looking at Figure 12, you can see that the fine-tuned horizontal offset is better overlaid than the approximated offset. The blended overlapping areas also do not show 'ghosts' in Figure 12 and Figure 13.

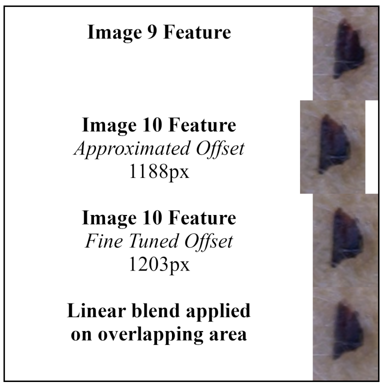

Fig. 12: Cropped feature of images with comparison between approximated and fine-tuned offsets, and final overlaid result

B. YOLO model configuration

The results of the 15 different model configuration tests can be seen in Table I per base model and image size combination. The finally chosen configuration is in bold, and inference times that go over 1s are in italics. As the average inference times increase with the larger models and image sizes, the standard deviation also increases for more unpredictable inference times.

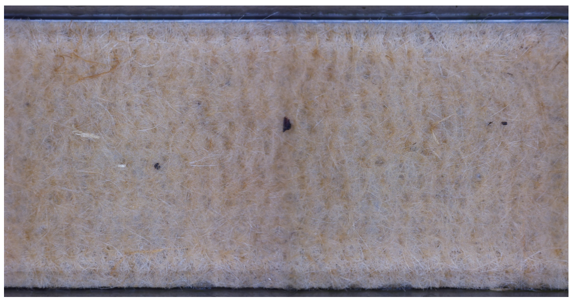

Fig. 13: Two sequential images of Figure 11 stitched together

Testing the average execution time of the additional software (all software without the YOLO inference) with different image sizes has the following results: 1280px takes 706.1ms (464.8ms of which is image processing), 960px takes 574.2ms, and 640px takes 342.9ms.

C. YOLO model format

Both the PyTorch (default) and NCNN (exported) formats have been tested on the Raspberry Pi 5 to validate the assumption that performance improves with NCNN and does not significantly alter precision. The metrics are the same as in the model training test. Table II shows the result of the comparison. The NCNN format outperforms the PyTorch format with an ∼22.6% average inference time. The accuracy scores are lower than the PyTorch alternative in most categories, but with the biggest and smallest differences only being -0.011 and 0.007, with a difference of -0.007 on average.

TABLE II: Comparison between YOLO model formats on the Raspberry Pi 5

Model Format		PyTorch	**NCNN**
Inference Time (ms)	Average	2040.8	**461.9**
	SD	35.83	**9.28**
mAP50	Average	**0.977**	0.974
	Bright & Spicy	**0.993**	0.990
	Amaranth	**0.962**	0.959
	Micro Thyme (Biostrate)	**0.982**	0.971
	Micro Thyme (Jute)	0.970	**0.977**

D. Single or multi-class model

The results of the comparison tests can be found in Table III. The inference times are relatively the same, with an average difference of only 29.2ms. The accuracy scores are mostly better or equal for the single-class model, with only one situation performing better for the multi-class model. The best-performing values are in bold.

TABLE III: Comparison between single class detection and multi-class classification on the Raspberry Pi 5

Classification Type		**Single class**	All classes
Inference Time (ms)	Average	461.9	**432.7**
	SD	**9.28**	12.01
mAP50	Average	**0.974**	0.973
	Bright & Spicy	0.990	0.990
	Amaranth	**0.959**	0.947
	Micro Thyme (Biostrate)	0.971	**0.991**
	Micro Thyme (Jute)	**0.977**	0.965

TABLE I: Inference time (ms) and mAP50 score of models for different training configurations

Base model	Image size	Inference time (ms)		mAP50				
		Average	SD	Average	Bright & Spicy	Amaranth	Micro Thyme (Biostrate)	Micro Thyme (Jute)
	640	113.8	29.16	0.911	0.994	0.967	0.852	0.832
Nano	960	251.0	5.97	0.968	0.991	0.953	0.968	0.961
	1280	**461.9**	**9.28**	**0.974**	**0.990**	**0.959**	**0.971**	**0.977**
	640	222.3	11.67	0.901	0.994	0.967	0.853	0.790
Small	960	523.5	8.49	0.963	0.990	0.953	0.953	0.956
	1280	968.9	16.46	0.978	0.990	0.965	0.982	0.976
	640	472.6	33.65	0.905	0.994	0.971	0.841	0.812
Medium	960	*1182.4*	84.37	0.949	0.989	0.95	0.938	0.918
	1280	*1968.7*	42.07	0.964	0.992	0.959	0.969	0.936
	640	830.5	12.93	0.907	0.993	0.967	0.871	0.798
Large	960	*2216.4*	23.84	0.967	0.992	0.967	0.962	0.945
	1280	*3970.2*	40.20	0.966	0.991	0.958	0.971	0.942
	640	*1317.9*	20.21	0.917	0.994	0.969	0.857	0.846
X-Large	960	*3186.2*	30.34	0.963	0.992	0.968	0.955	0.936
	1280	*5390.3*	77.92	0.967	0.989	0.968	0.977	0.935

E. Generalisation Test

Table IV shows the results of the generalisation test, where a model is trained without jute data and trained further with different amounts of jute data.

The inference times between the different models are relatively the same, with the standard deviation increasing slightly for additional data. The accuracy scores differ only slightly for the Bright & Spicy seeds and Amaranth seeds, but both the Micro Thyme seed situations see a major difference between the introduced data compared to the base model. With the biostrate, the accuracy lowers at first with a small amount of data introduced but eventually returns to a similar score with 4 jute substrate images. For the Micro Thyme seeds on a jute substrate, the score sharply increases for the first introduced data, and afterwards only increases to its best score with the addition of 4 jute substrate images. The average mAP50 score is the best at 4 images due to the large improvement for images with Micro Thyme seeds on a jute substrate.

F. Seed duplicate detection

The annotated images can be found in Figure 14. In the overlapping regions of the images, there are 28 seeds fully visible with 7 partially obscured. As seen in Figure 14d, only one fully visible seed is not detected in both images, and two fully visible seeds are only detected in one image. The rest of the fully visible seeds have detections in both images and are correctly merged into one seed with the average position and size. The partially visible seeds mostly have only one detection, and therefore are not merged with other seeds.

G. Accuracy of system

A portion of the combined gutter image and the corresponding density map visualisation (colourmap) are displayed in Figure 15 and 18, respectively. The full images are in Appendix A. The saved metadata JSON file has all data points present together with all configuration and metadata, but since the metadata generated by the current prototype software is 35,989 lines long, this report will not include the entire file. Instead, only the validation metrics derived from the metadata are presented: Seed Count is 2790, Average Density is 0.01116, Minimum Local Density is -1.99×10^{-16}, and Maximum Local Density is 0.0278. All densities are in seeds/mm^2. The average execution time between images is 1238.8ms.

After manually correcting the seed detections for the 'ground truth', the following data was calculated: The analysed gutter had 2811 total seeds, with the prototype QA station having 94 false positives and 115 false negatives. This means the QA station prototype's recall and precision are 0.9591 and 0.9663, respectively, with an F1 score of 0.9627. The 'ground truth' average density is 0.011244, the minimum local density is 0.0, and the maximum local density is 0.01477.

The combined image of the gutter in Appendix A also shows artefacts/'ghosts' in the blended regions where two images are stitched together.

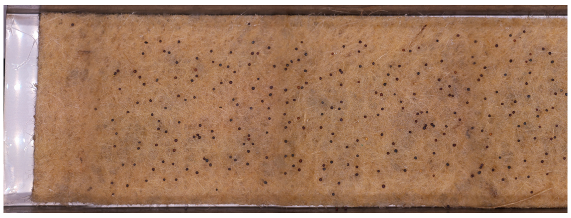
Fig. 15: First 300mm of the combined image of the seeded gutter validated in the system test from stitched-together images. Full figure in Appendix A.

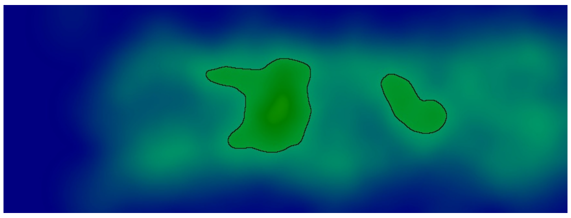
Fig. 16: First 300mm of the density colourmap of the seeded gutter validated in the system test. Full figure in Appendix A.

VII. Discussion

A. Image processing

Looking at Figure 12, a glance can tell that the approximated offset of 1188px for image 10 does not align very well with the same feature in image 9. With the fine-tuned offset, however, little improvement can be gained. If done manually, the offset might differ a few pixels in either direction, but the fine-tuned offset takes the whole image into account and not just the focussed feature. The blended feature from both images in Figure 12 and Figure 13 also shows no 'ghosting' or issues with the overlapping of the images.

The image processing, therefore, performs its goal of combining and finding the offset between captured images.

TABLE IV: Inference time (ms) and mAP50 score for different amounts of training data of jute substrates

Nr. of jute images	Inference time (ms)		mAP50				
	Average	SD	Average	Bright & Spicy	Amaranth	Micro Thyme (Biostrate)	Micro Thyme (Jute)
0	433.7	**8.57**	0.769	0.993	0.959	**0.992**	0.132
1	469.2	10.04	0.977	0.993	**0.962**	0.982	0.972
2	437.5	10.49	0.975	0.993	0.956	0.981	0.970
3	**432.0**	13.48	0.974	0.993	0.959	0.982	0.963
4	444.9	13.31	**0.980**	0.993	0.956	0.99	**0.981**

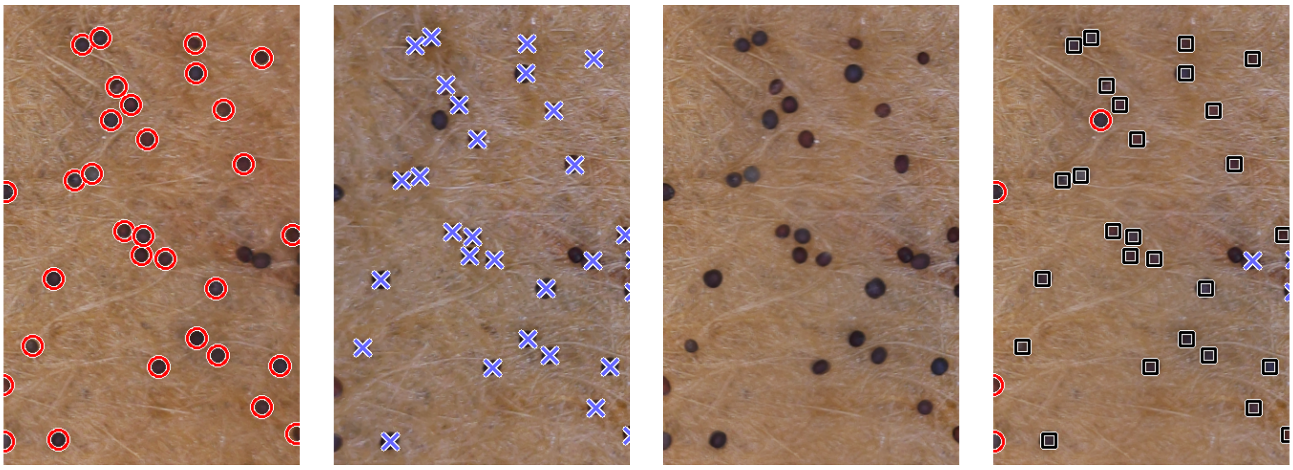
(a) Image 1 detections **(b)** Image 2 detections **(c)** Linear blended images **(d)** Merged seeds

Fig. 14: Visualisation of seed detections from two images captured in sequence (14a and 14b), and the merged result (14d). Red circles show detections only in 14a, blue crosses are detections only in 14b, and black squared are detections from both images that have merged into one. For clarity, a cropped overlapping section of the images is used instead of the whole image.

B. *YOLO model configuration*

Using the gathered information, some observations can be made. First, the base model size has little effect on the resulting precision while dramatically increasing the inference time. This is assumed to be because the objects in this project are limited to very basic shapes and do not need complex models with many high-level features to detect them. Increasing the image size has a positive effect on precision, with 1280px reaching the best average scores for all base models, but it non-linearly increases the inference time and execution time. The small seed (Micro Thyme) situations mostly contribute to the increase in precision for bigger image sizes; as presumably mentioned in related works, the objects are too small for consistent detection.

For this reason, the 1280px image size was chosen for the final configuration, and this just leaves the choice of the base model. Since the base model size almost has no impact on the precision, the choice was made to use the nano base model for this project as it is below the inference limit with a relatively consistent inference time and gives the most performance headroom for the rest of the software execution time. The average combined execution time for the software at 1280px with a nano base model is 1168ms. This means that the total execution time headroom is 832ms from these tests and can potentially be used with the QA station for a faster conveyor speed than 50mm/s.

C. *YOLO model format*

While the NCNN format drastically lowers the inference time of the model, the precision is slightly lower. This change, however, is insignificant enough to keep using the NCNN format for the project, as the largest difference is -0.011 mAP50 for small seeds on biostrate, with the difference only being -0.003 for the average mAP50 score (-0.3%), which was expected from initial tests. This difference in accuracy is likely because of rounding errors after each layer and variable types used in the model calculations.

Because the software needs to run in real-time, the assumption and use of the NCNN format have been validated.

D. *Single or multi-class model*

The model is more generic with fused classes, with multi-class comparatively having significantly better and worse classes while the average precision is almost the same. It is also clear that the multi-class model has even or higher scores in the 'easy' types, with the dark seeds on a white background types in this case and the light seeds on a white background or small dark seeds on a noisy jute substrate being worse. Surprisingly, the multi-class one has better inference time, but no clear reasoning can be given as to why, except for the possibilities that the implementation by Ultralytics is better suited for multi-class models or that there was a slight inconsistency in the Raspberry Pi's background CPU usage during testing.

Since the project's goal is to make a very generic model that can easily detect seeds in new situations with later possible fine-tuning, the single-class method is indeed the best choice. The performance difference is also not big enough to impact this decision, as using the smallest model size has given enough performance headroom.

E. *Generalisation Test*

With no jute data, the model is, as expected, not accurate for any images with jute as the substrate. Training the model further with only one image for training and one for validation, the accuracy of jute substrates immediately comes close to the other situations in the dataset with a 0.972 mAP50 score. Introducing more jute data into the dataset does raise the accuracy of the model further, to an average of 0.980 mAP50 with four jute images in the training dataset. From these results, it looks like manually labelling two images of a new substrate, one for training and one for validation, is enough to get initial accurate predictions for further assisted labelling for the new substrate. This additional data gathered with assisted labelling can then be used for further improvements in accuracy.

While these test results show that the model is not general enough to immediately work with a new background/substrate, they do prove that little work is needed to get better results. The hope is that with more different seeds and substrates, the model will become much more generic.

F. Seed duplicate detection

From the results in Figure 14, it appears that the detection and merging of duplicate seeds perform as expected while also validating the conversion of relative seed coordinates to absolute coordinates in the seeded gutter. No duplicate seeds can be found in the software's output, and while some seeds only have one detection and one is not detected at all, this is not an issue with this software component. If anything, it proves that the software does not interfere with seeds with a single detection or introduces additional detections.

G. Accuracy of system

When looking at the full image of the seeded gutter in Appendix A, stitching artefacts can be seen in the blended regions where two images are stitched together. These issues are caused by two main factors: vibrations in the seeding machine and poor calibration during preparation. For this project, the vibrations of the seeding machine are an external problem that Growy will address later but will stay present for the duration of the project. The vibrations are caused by pumps placed on the seeding machine that regularly switch on to regulate the water pressure. They shake the camera during image acquisition, distorting the images. The poor calibration was, unfortunately, caused by hasty preparation and a sub-optimal calibration grid.

These two factors caused two sequential images to not line up perfectly during the image stitching steps and, therefore, caused the seed duplication detection not to remove all duplicate seeds. This improper image stitching caused multiple false positives regarding system performance. Analysing the results, 93 of the 94 false positives occur at the image seams. subsection VI-A and VI-F show that the individual software responsible for the transformation and removal of duplicates works when calibrated correctly, so for this discussion, the recall, precision and F1 scores are also calculated when ignoring the 93 duplicates from the vibrations and calibration issues. This results in improved precision and F1 scores of 0.9996 and 0.9789, respectively. These scores show that the current prototype system has a much higher hypothetical performance if the vibration issue is eventually resolved and better calibration is done.

The results of this test also fulfil the requirements regarding data logging, config metadata, minimum recall and precision scores, seed positions, density metrics, and a density colourmap. The execution is also far below the limit of 2s (1238.8ms), with enough headroom for future iterations.

VIII. Conclusion

With the project concluded, a conclusion can be made on the project's success as a whole. The objective of the QA station prototype is to show a viable design for automating the validation and inspection currently done by human operators while providing data points for reporting and analysis. The requirements in section IV were made to prove and validate this design.

They mention that the QA station should be capable of real-time analysis of 2.5m seeded gutters, ensuring compatibility with various seed and substrate combinations. The prototype was required to integrate with the seeding machine for data exchange and provide detailed outputs for validation of the gutters and logging, including stitched images, seed counts, positions, density metrics, and metadata.

The QA system prototype achieves most of these objectives as demonstrated by the system tests in section VI. It successfully analyses the seeded gutters in real-time, maintaining execution times far below the 2-second threshold because of the relatively small YOLO base model. It operates effectively across various seed and substrate combinations using a deep-learning model to generalise seed detection. It generates data points with a complete high-resolution image representing the gutter by stitching multiple smaller images together. The data points include detailed seed counts and positions, density metrics with visualisation, and system configuration data. This information is accurately logged for reporting and analysis by saving a JSON file with all data points to an online database. Additionally, the system achieves seed detection recall and precision scores exceeding the 0.95 requirement by using the YOLO object detection.

The part of this project and prototype that fails to meet the requirements is the integration. This stems from the issue that the QA station has not had the chance to be integrated with the seeding line during the project and could, therefore, not be used yet in actual production to try out the designed integration for validation in production, if only as a tool for a human operator. While the prototype has not been integrated, preparations have been made by emulation, and it is expected that the integration will not cause any issues and can be done with relatively little time and effort.

While the calculated data points like the density map provide valuable information and fulfil the requirements, the extracted minimum and maximum densities from the density map, for example, are still potentially lacking in being fully used as validation metrics. These metrics do not, for example, consider how much area is over the threshold, and the minimum density in testing always lies close to the border of the gutters instead of giving important information on the seeds in the rest of the gutter. This uncertainty of how useful the chosen data points are comes from the fact that Growy has not yet used automation to validate the seeding machine and does not know yet what data points they will eventually use. It is assumed that the current data points are sufficient to prove the viability of automating the validation, but later versions will likely use more sophisticated data points.

Using a Raspberry Pi 5 as the platform for this project did not hinder the performance or accuracy of the prototype in hindsight and should suffice in future iterations. Growy might want to switch to another platform for external reasons or because of a change in requirements for the QA station. One future limitation of the Raspberry Pi 5 could be that Growy will eventually want to increase the conveyor speed of the seeding line, making the interval between image captures shorter than the current execution time, demanding higher-performing hardware.

One factor that hindered this project, if only slightly, was that the seeding line was a constantly used production machine, so modifications to the prototype hardware could not be instantly made. Also, the seeding line did not use the jute substrate until late in the project, which made the earlier choices somewhat uncertain. Fortunately, it was possible to work around these issues by planning accordingly.

Since no literature studies were done on using image processing or neural networks to detect different seed types on noisy backgrounds without the need for classifying them and combining multiple smaller images to combine the results, this project and the prototype showed that this method works. Especially compared to a human operator, the prototype gives valuable data for evaluated gutters that can be used to validate the gutters better and analyse the results for potential increased yield.

IX. Future Improvements and Research

Some additional improvements or research could be made to improve future iterations of the automated QA station.

The most obvious one is to integrate the QA station with the seeding line to see how it performs in the intended environment with the seeding line. This way, more performance data can be gathered, and unknown issues or obstacles can be found.

Improving the density metrics used for validation should also require additional research to provide more useful and reliable metrics to validate the seeded gutters. This would have to be done with the data team at Growy to discuss how a gutter would be validated. One idea is to use local minima instead of the global minimum of the density map as a validation metric, as this could better represent seed distribution by focusing on local minima away from the edges of the map.

Testing the QA station with more seed types with more extreme sizes, shapes, or colours could also give more insight into how well the seed detection generalises the seed types and performs with new extreme seed types. Using more data for the already used seed types could also potentially increase performance, so increasing the dataset with existing and new seed types is an interesting opportunity to try and improve the current prototype.

During this project, Ultralytics published their new models, YOLO11[46], that have the potential to increase accuracy and precision while having the same performance for inference. Using these new models could be interesting for additional improvements to the QA station in terms of accuracy and precision of seed detection.

X. Acknowledgements

Thanks to Nikolaos Alachiotis, Sebastian Bunda, Amirreza Yousefzadeh, and Luuk Spreeuwers from the University of Twente, and Frank Ridder, Jochem Meuwese, Mert Imre, and others at Growy for helping me with the project. Your feedback, assistance, and time are greatly appreciated.

References

[1] D. Egli, "Plant density and soybean yield," *Crop Science*, vol. 28, no. 6, pp. 977–981, 1988.

[2] Ultralytics Inc, "NEW - YOLOv8 in PyTorch," https://github.com/ultralytics/ultralytics, 2024, accessed: (May 14th 2024).

[3] CANopen.nl, "Welkom bij CANopen.nl," https://www.canopen.nl/, 2024, accessed: (October 9th 2024).

[4] A. Bhargava and A. Bansal, "Fruits and vegetables quality evaluation using computer vision: A review," *Journal of King Saud University-Computer and Information Sciences*, vol. 33, no. 3, pp. 243–257, 2021.

[5] F. A. Padhilah and W. Wahab, "Comparison between image processing methods for detecting object like circle," in *Proceedings of the 2018 International Conference on Digital Medicine and Image Processing*, 2018, pp. 33–36.

[6] R. Maini and H. Aggarwal, "Study and comparison of various image edge detection techniques," *International journal of image processing (IJIP)*, vol. 3, no. 1, pp. 1–11, 2009.

[7] S. Gebhardt, J. Schellberg, R. Lock, and W. Kühbauch, "Identification of broad-leaved dock (rumex obtusifolius l.) on grassland by means of digital image processing," *Precision Agriculture*, vol. 7, pp. 165–178, 2006.

[8] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.

[9] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3523–3542, 2021.

[10] S. Gebhardt and W. Kühbauch, "A new algorithm for automatic rumex obtusifolius detection in digital images using colour and texture features and the influence of image resolution," *Precision Agriculture*, vol. 8, pp. 1–13, 2007.

[11] G. Shrivakshan and C. Chandrasekar, "A comparison of various edge detection techniques used in image processing," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 5, p. 269, 2012.

[12] C. Akinlar and C. Topal, "Edcircles: A real-time circle detector with a false detection control," *Pattern Recognition*, vol. 46, no. 3, pp. 725–740, 2013.

[13] Y. Gulzar, Y. Hamid, A. B. Soomro, A. A. Alwan, and L. Journaux, "A convolution neural network-based seed classification system," *Symmetry*, vol. 12, no. 12, p. 2018, 2020.

[14] F. Kurtulmuş, İ. Alibaş, and I. Kavdır, "Classification of pepper seeds using machine vision based on neural network," *International Journal of Agricultural and Biological Engineering*, vol. 9, no. 1, pp. 51–62, 2016.

[15] L. Wang, J. Liu, J. Zhang, J. Wang, and X. Fan, "Corn seed defect detection based on watershed algorithm and two-pathway convolutional neural networks," *Frontiers in Plant Science*, vol. 13, p. 730190, 2022.

[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[19] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[20] Y. Xiao, Z. Tian, J. Yu, Y. Zhang, S. Liu, S. Du, and X. Lan, "A review of object detection based on deep learning," *Multimedia Tools and Applications*, vol. 79, pp. 23 729–23 791, 2020.

[21] L.-D. Quach, K. N. Quoc, A. N. Quynh, and H. T. Ngoc, "Evaluating the effectiveness of yolo models in different sized object detection and feature-based classification of small objects," *Journal of Advances in Information Technology*, vol. 14, no. 5, pp. 907–917, 2023.

[22] A. Dhillon and G. K. Verma, "Convolutional neural network: a review of models, methodologies and applications to object detection," *Progress in Artificial Intelligence*, vol. 9, no. 2, pp. 85–112, 2020.

[23] N. Genze, R. Bharti, M. Grieb, S. J. Schultheiss, and D. G. Grimm, "Accurate machine learning-based germination detection, prediction and quality assessment of three grain crops," *Plant methods*, vol. 16, pp. 1–11, 2020.

[24] Y. Guo, S. Li, Z. Zhang, Y. Li, Z. Hu, D. Xin, Q. Chen, J. Wang, and R. Zhu, "Automatic and accurate calculation of rice seed setting rate based on image segmentation and deep learning," *Frontiers in plant science*, vol. 12, p. 770916, 2021.

[25] B. Jabir, N. Falih, and K. Rahmani, "Accuracy and efficiency comparison of object detection open-source models." *International Journal of Online & Biomedical Engineering*, vol. 17, no. 5, 2021.

[26] B. Gomathy, N. Sengottaiyan, P. Thirumoorthy, P. Kalyanasundaram *et al.*, "Performance comparison of object detection neural network models based on accuracy and latency," in *2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*. IEEE, 2024, pp. 1040–1044.

[27] Y. Hamid, S. Wani, A. B. Soomro, A. A. Alwan, and Y. Gulzar, "Smart seed classification system based on mobilenetv2 architecture," in *2022 2nd International Conference on Computing and Information Technology (ICCIT)*. IEEE, 2022, pp. 217–222.

[28] J. Qiao, Y. Liao, C. Yin, X. Yang, H. M. Tú, W. Wang, and Y. Liu, "Vigour testing for the rice seed with computer vision-based techniques," *Frontiers in Plant Science*, vol. 14, p. 1194701, 2023.

[29] K. Sabanci, M. F. Aslan, E. Ropelewska, and M. F. Unlersen, "A convolutional neural network-based comparative study for pepper seed classification: Analysis of selected deep features with support vector machine," *Journal of Food Process Engineering*, vol. 45, no. 6, p. e13955, 2022.

[30] T. Luo, J. Zhao, Y. Gu, S. Zhang, X. Qiao, W. Tian, and Y. Han, "Classification of weed seeds based on visual images and deep learning," *Information Processing in Agriculture*, vol. 10, no. 1, pp. 40–51, 2023.

[31] S. Javanmardi, S.-H. M. Ashtiani, F. J. Verbeek, and A. Martynenko, "Computer-vision classification of corn seed varieties using deep convolutional neural network," *Journal of Stored Products Research*, vol. 92, p. 101800, 2021.

[32] L. Pang, S. Men, L. Yan, and J. Xiao, "Rapid vitality estimation and prediction of corn seeds based on spectra and images using deep learning and hyperspectral imaging techniques," *Ieee Access*, vol. 8, pp. 123 026–123 036, 2020.

[33] K. Kiratiratanapruk, P. Temniranrat, W. Sinthupinyo, P. Prempree, K. Chaitavon, S. Porntheeraphat, A. Prasertsak *et al.*, "Development of paddy rice seed classification process using machine learning techniques for automatic grading machine," *Journal of Sensors*, vol. 2020, 2020.

[34] Y. Jiang, X. Gong, D. Liu, Y. Cheng, C. Fang, X. Shen, J. Yang, P. Zhou, and Z. Wang, "Enlightengan: Deep light enhancement without paired supervision," *IEEE transactions on image processing*, vol. 30, pp. 2340–2349, 2021.

[35] C. Wei, W. Wang, W. Yang, and J. Liu, "Deep retinex decomposi-

tion for low-light enhancement," *arXiv preprint arXiv:1808.04560*, 2018.

[36] M. Rad, P. M. Roth, and V. Lepetit, "Alcn: Adaptive local contrast normalization for robust object detection and 3d pose estimation," *BMVC 2017*, 2017.

[37] Z. Wang and Z. Yang, "Review on image-stitching techniques," *Multimedia Systems*, vol. 26, no. 4, pp. 413–430, 2020.

[38] N. Yan, Y. Mei, L. Xu, H. Yu, B. Sun, Z. Wang, and Y. Chen, "Deep learning on image stitching with multi-viewpoint images: A survey," *Neural Processing Letters*, vol. 55, no. 4, pp. 3863–3898, 2023.

[39] F. C. Akyon, S. O. Altinuc, and A. Temizel, "Slicing aided hyper inference and fine-tuning for small object detection," in *2022 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2022, pp. 966–970.

[40] Ultralytics Inc, "Raspberry Pi - Ultralytics YOLO Docs," https://docs.ultralytics.com/guides/raspberry-pi/, 2024, accessed: (October 7th 2024).

[41] Raspberry Pi Ltd, "Camera - Raspberry Pi Documentation," https://www.raspberrypi.com/documentation/accessories/camera.html, 2024, accessed: (May 14th 2024).

[42] Common Objects in Context, "COCO - Common Objects in Context," https://cocodataset.org/, 2024, accessed: (May 14th 2024).

[43] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," 2019. [Online]. Available: https://arxiv.org/abs/1902.09630

[44] Lely CANopen, "Lely CANopen - Lely CANopen," https://opensource.lely.com/canopen/, 2024, accessed: (October 9th 2024).

[45] Christian Sandberg, "CANopen for Python," https://opensource.lely.com/canopen/, 2024, accessed: (October 9th 2024).

[46] Ultralytics Inc, "YOLO11 - Ultralytics YOLO Docs," https://docs.ultralytics.com/models/yolo11/, 2024, accessed: (October 9th 2024).

Appendix A
Full combined image and colourmap of system test

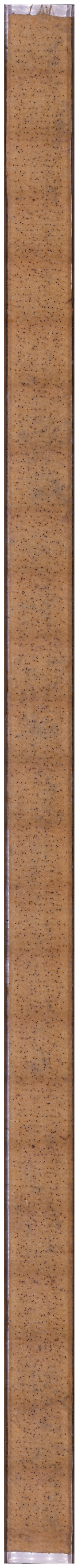

Fig. 17: The combined image of the seeded gutter validated in the system test from stitched-together images

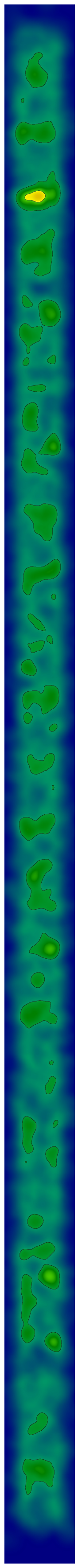

Fig. 18: The density colourmap of the seeded gutter validated in the system test

APPENDIX B
PSUEDOCODE

Algorithm 1 Image Stitching Pseudocode

```
approxOffset = conveyorSpeed * captureInterval

bestOffset = approxOffset
leastDifference = 1.0
for offset from approxOffset-range to approxOffset+range do
    difference = Sum(Abs(img1[offset:] - img2[:imgSize-offset]))
    if difference > leastDifference then
        leastDifference = difference
        bestOffset = offset
    end if
end for
absoluteOffset = RelativeOffsetToAbsolute(bestOffset)

overlapSize = gutterWidth - absoluteOffset
gutterMask = LinearGadientMask(overlapSize, 'leftToRight')
frameMask = LinearGadientMask(overlapSize, 'rightToLeft')

originalGutter = gutterImg[:absoluteOffset]
blendedOverlap = gutterImg[absoluteOffset:]*gutterMask + newImg[:overlapSize]*frameMask
leftoverImage = newImg[overlapSize:]

gutterImg = Concatenate(originalGutter, blendedOverlap, leftoverImage)
```

Algorithm 2 Absolute seed position Pseudocode

```
imageSeeds = AddHorizontal(imageSeeds, horizontalOffset)
overlapStart = horizontalOffset
overlapStop = gutterLength

for gSeed in gutterSeeds do
    for iSeed in imageSeeds do
        if (gSeed.x or iSeed.x < overlapStart) or (gSeed.x or iSeed.x > overlapStop) then
            continue
        if IntersectionOverUnion(gSeed, iSeed) < IouThreshold then
            continue

        gSeed = MergeSeeds(gSeed, iSeed)
        imageSeeds.remove(iSeed)
    end for
end for
gutterSeeds += imageSeeds
```