Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Analysis of time-lapse images for high-throughput plant phenotyping

Thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

MARCUS SCHUTTE

Delft, the Netherlands
July 2020

BSc Thesis APPLIED MATHEMATICS

"Analysis of time-lapse images for high-throughput plant phenotyping"

MARCUS SCHUTTE

Delft University of Technology

Supervisor

Dr. N.V. Budko

Further committee members

Dr. ir. W.T. van Horssen

July, 2020 Delft

Contents

1. Introduction

With an increasing human population, it becomes ever more challenging to meet the demand for food. In agriculture, different plant phenotypes have different traits which makes certain phenotypes produce better crop quantity and quality. Different environmental circumstances influence the plant growth and the crop yield as well. Time-lapse images of growing plants allow for a high-throughput phenotyping and analysis of traits for subsequent selective breeding.

In this Thesis several algorithms are proposed to process the data set acquired in the Flight to Vitality (FtV) project. As part of the FtV project, a large number of pots containing potato tubers is placed in a grid pattern upon 8 separate tables. The tables themselves were placed in two climate-controlled rooms where four different environmental conditions were maintained by controlling the room temperature and local humidity. Each individual pot on the table had to be identified and cut out of the image. This has been achieved with the use of image thresholding and exploiting the regularities in the pot placement and image acquisition. The leaf canopy was filtered out of the images with an adaptive HSV-based green-detection algorithm. The resulting canopy data is analysed and compared to a population growth model. The used growth model is based on a large Ordinary Differential Equation (ODE) system transformed by a two-dimensional conservation law. The two dimensions are: phase-space of canopy sizes and growth rates. Finally, the height of the plants is estimated from several stereoscopic images using the methods of photogrammetry.

2. Data pre-processing

The data set of the Flight to Vitality (FtV) project contains time-lapse images of potato plants in pots. The data is gathered for the purpose of analysing the growth of the plants. The plants were photographed from above. Every image contains 8 rows and 10 columns of plants. In order to get an estimate of the height of the plant, there were three pictures taken depicting each plant from different angles. The data set contains the images taken from pots that are placed on eight different tables. Of each table with pots on it, there were 27 pictures taken, with the whole acquisition procedure repeated over multiple days.

To be able to use this data set, it is of importance to pre-process the raw data, i.e., the separate overlapping images, to such a form that they can be analysed. Firstly, the individual pots in the photographs need to be labelled and cut out of the image. Since every pot is in three images, the images depicting the same pot have to be grouped together. An image thresholding method is used to cut out individual plant pots from the photographs. For grouping the images, the map that was included with the data set is used.

2.1. **Image filters for pot detection.**

The FtV data set is a specific data set for this project. Therefore, this chapter will be rather intuitive and only partially based on previous research. The goal is to find a method that can cut the pots out of the images in a stable way. To cut out the pots from the images the edges of the pots need to be found. To find the edges of the pots a commonly used edge detection method described in the book on computer vision by E.R. Davies (2017) [1] will be applied. Specifically, image thresholding, Laplace edge operators and Canny edge detector were applied. It was found that the image thresholding was the most suitable for this study.

2.1.1. *Thresholding.*

Image thresholding is one of the most basic filter techniques. Modern images are stored as maps of RGB-values, where the red, green and blue values are stored per pixel as a integer between 0 and 255. For image thresholding a certain threshold needs to be determined to compare the pixel values of the image with. A threshold value can be determined by hand by determining in which colour range the object of interest appears. A threshold value can also be determined from the images themselves, by taking the mean of the RGB-values, for instance. Filtering an image with a threshold value results in a binary image where 1 is assigned to a pixel that satisfies the threshold condition and 0 otherwise.

$$I(u,v) = \begin{cases} 1 & r_b < R(u,v) < r_a \text{ and } g_b G < (u,v) < g_a \text{ and } , b_b < B(u,v) < b_a \\ 0 & \text{else} \end{cases}$$

Here the $r_b,\ r_a,\ g_b,\ g_a,\ b_b$ and b_a are the RGB-values above and below threshold respectively. $I(u,v)$ is the intensity of the filtered image at pixel (u,v). $R(u,v), B(u,v)$ and $G(u,v)$ are the colors of the pixel at (u,v)

When applying Image Thresholding (and other image filter techniques) it is common to convert the color image to a grayscale image. The most basic method to convert a color image to grayscale is linear conversion. The grayscale values are scaled to values between 1 and 0, where 1 is white and 0 represents dark or black.

$$I(u,v) := c_r R(u,v)/255 + c_g G(u,v)/255 + c_b B(u,v)/255$$

where I(u,v) is the pixel intensity. $c_r + c_g + c_b \in [0,1]$ such that $c_r + c_g + c_b = 1$. This will cause the grayscale values to be between 0 and 1. In practice the constants that are used most often are $c_r = 0.2126,\ c_g = 0.7152$ and $c_b = 0.0722$, according to Wikipedia [2].

For thresholding the gray scale image, just a single threshold needs to be determined. The thresholding happens via the same principle:

$$I_{filtered}(u,v) = \begin{cases} 1 & d_b < I(u,v) d_a \\ 0 & \text{else} \end{cases}$$

2.1.2. *Laplace filter.*

There is a whole class of filters based on differential equations. The Laplace filter is based on the Laplace operator and the notion that edges are often on places where there is a high discrepancy in pixel intensity.

In continuous form, the Laplace operator is written as follows:

$$\Delta^2 = \frac{\delta^2}{\delta x^2} + \frac{\delta^2}{\delta y^2}.$$

The discrete Laplace operator takes the following form:

$$\Delta^2 I(u,v) = \frac{I(u-1,v) - 2I(u,v) + u(u+1,v}{\Delta x^2} + \frac{I(u,v-1) - 2I(u,v) + I(u,v+1}{\Delta y^2},$$

where we can take $\Delta x = \Delta y = 1$, i.e., we set the distance between two pixels to be equal to 1:

$$\Delta^2 I(u,v) = I(u-1,v) + I(u,v-1) - 4I(u,v) + I(u+1,v) + I(u,v+1).$$

For filtering a grayscale image with the Laplace filter, it is more convenient to write the filter as a mask in matrix form:

$$M = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

The matrix form is more convenient, because it makes it possible to take convolution of the image and the mask. If $I(u,v)$ is the image intensity in gray scale value at pixel (u,v), then it is given by

$$I_f(u,v) := \sum_{i=1}^{3} \sum_{j=1}^{3} M_{i,j} I(u+i-1, v+i-1).$$

A way to visualise this, is to imagine the mask sliding over the image. This concept can be generalized for larger than 3 by 3 matrices. At the boundary, the image is often reflected.

Because the Laplace operator is very sensitive for noise, the Laplace operator is further combined with the Gaussian noise filter. The result is the mask below, which is the filter that is used in practise whenever a reference to the Laplace filter is made:

$$M_{Laplace} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}.$$

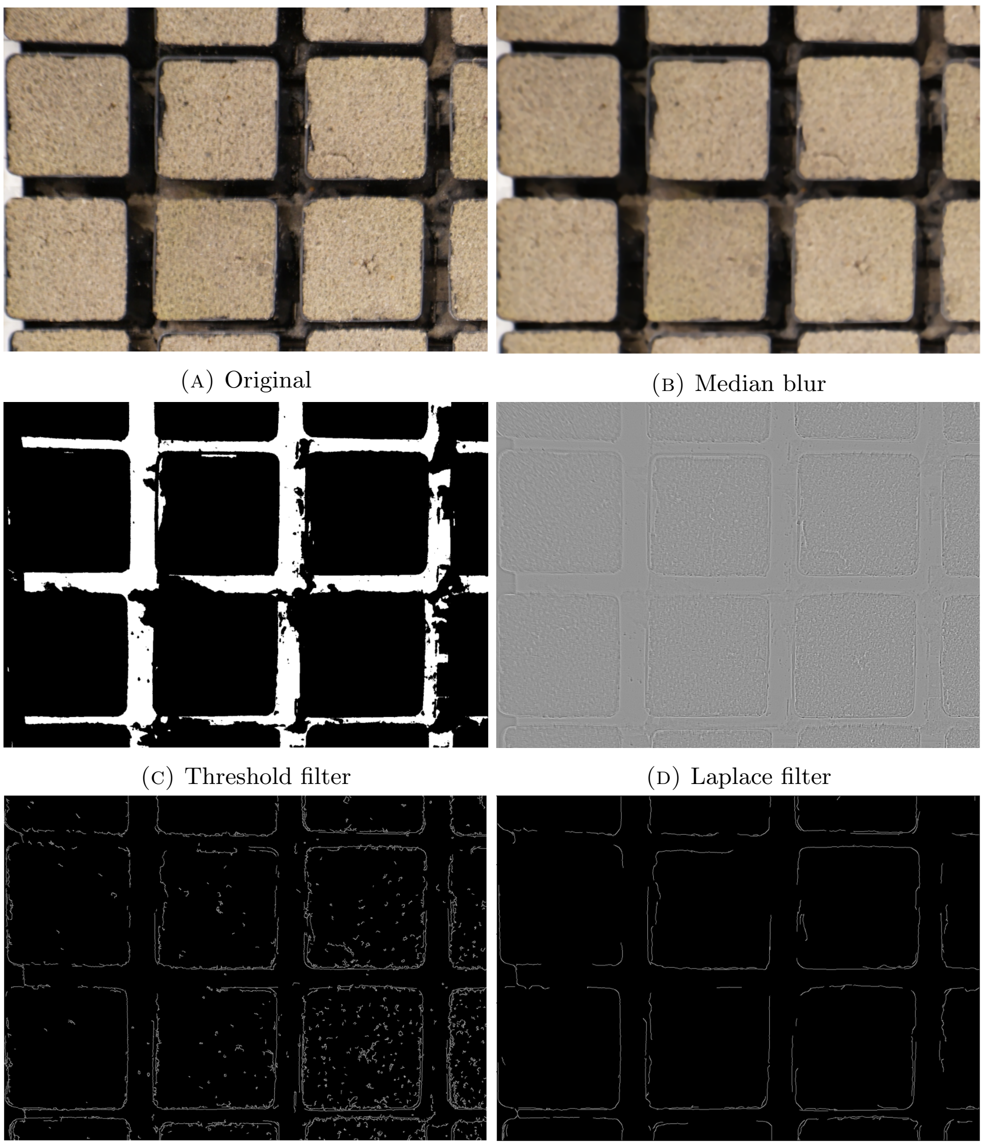
2.1.3. *Canny edge detection.* According to the book: "Computer and machine vision" of E.R. Davies p. 136 [1], the Canny edge detector is currently one of the most used edge detection algorithms. An explanation of this algorithm is beyond the scope of this paper. The Canny edge filter processes the images and returns a binary image where all edges are 1 pixel in width. This is the reason that this filter is considered as especially useful.

2.2. **Application and tuning of image filters.** The filters described above are tested on different pictures of the data set with the aim to find a robust method to cut out the images of pots from the pictures.

Figure 1 illustrates the application of the filters to a part of an image from the data set. In Figure 1a the original image is shown. To reduce noise in the output, the image is pre-processed. Figure 1b shows the results of pre-processing with a median blur. The median blur evens out extreme values by replacing the pixel value at a target pixel by the median value of the pixel neighborhood. In Figure 1c a threshold filter is applied to the original image. In Figure 1d the Laplace filter is applied. The Canny edge detector applied on the original, Figure 1e, results in an abundance of noise in the output image. To ensure the outcomes are more smooth the image is pre-processed with a median blur. However, this causes the result to lose some indicators of the boundaries of the pots as well.

The images in Figure 1 show that the threshold filter gives an indication of the pot's edges. The Laplace filter only gives a very weak indication of the boundary. The Canny edge detection applied to the original image is very noisy. When it is applied to the pre-processed image however, it shows a boundary that is sufficiently pronounced, although the pre-processing also caused loss of boundary fragments.

Since the threshold filter distinguishes the boundaries of the pot excellently, this filter is chosen to be the most appropriate for our purposes. The Canny edge detector filters the boundary sufficiently as well, albeit after pre-processing the image. However Canny edge detection in combination with a median blur is a time-consuming algorithm, while the threshold filter requires only one single comparison per pixel. Therefore the threshold filter is chosen as most suitable.



(A) Original (B) Median blur

(C) Threshold filter (D) Laplace filter

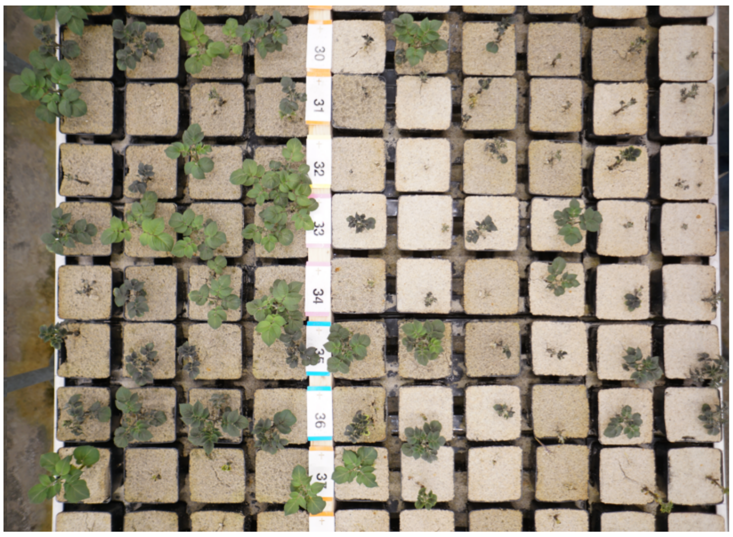
(E) Canny edge detection (F) Median blur + Canny edge detection

FIGURE 1. different filters applied on original Image

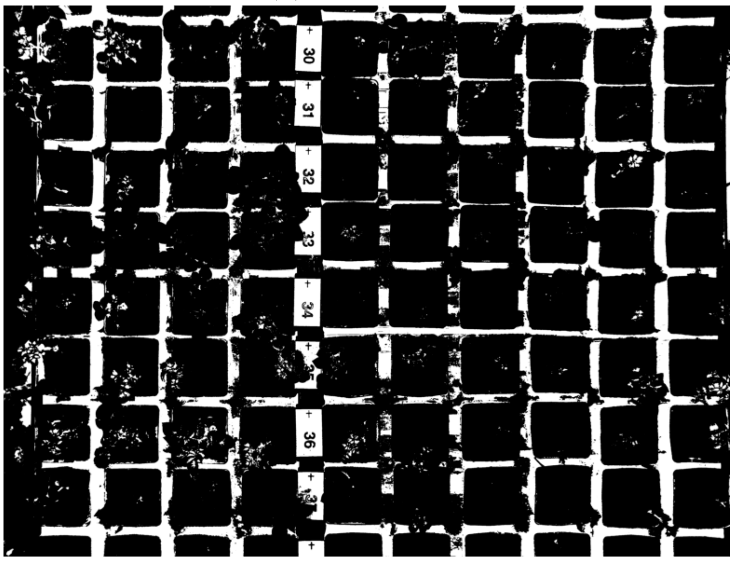
2.2.1. *Adjusting the threshold filter.* Because some pictures have different brightness then others, the threshold value that is used for Thresholding is based on the mean of the pixel values. In this case 0.7 times the mean is used as upper bound. Every pixel with a value larger than 0.7 is considered to be on the foreground. The other pixels are background, hence the edges between the pots. By testing this filter and by inspecting the data it was observed (see figure 2a) that the brightness in the middle of the table is higher than at the edges (this can be explained by the lamps hanging centred above the table). As a result thresholding against one threshold was not always sufficient. In order to correct this, the image is split horizontally into five parts. These parts are then processed separately.

2.3. **Cutting out individual pots.** The images are now filtered in order to locate the edges of the pots. The filtered images are binary maps with ones between the pots and zeros inside the pots. In this Section a method will be discussed which cuts out a pot assuming that an initial location inside the pot is given.

To find the boundary of a pot, for a starting point there is a initial point needed. Note that this point is not necessarily the middle of the pot. By scanning the image in horizontal and vertical direction, the boundary of the pot can be found. To prevent noise from effecting the results, a band parallel to the boundary of interest is positioned over the image, see figure 3b). The pixel values in this band are added up. If the sum is greater than a certain threshold (here half area of the band is used) the location of the

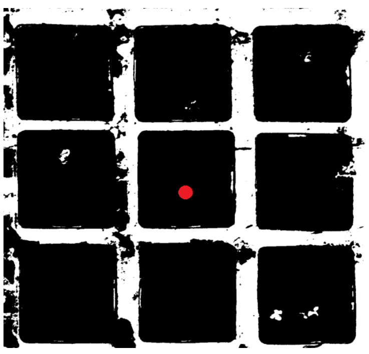


(A) original image

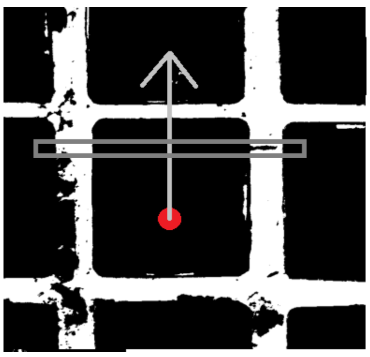


(B) threshold filter

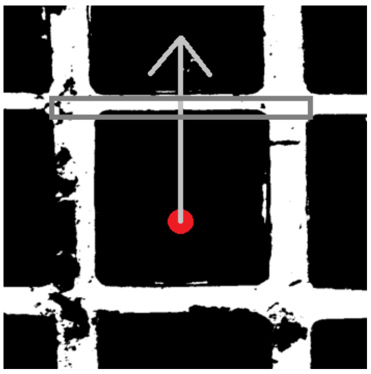
FIGURE 2. Composite image of thresholding applied to five separate slices due to a difference of light intensity



(A) Initial point



(B) Finding boundaries



(C) Finding edges

FIGURE 3

band is stored in a list. Then, the mean value of this list is calculated. The result will be roughly the middle of the pots boundary.

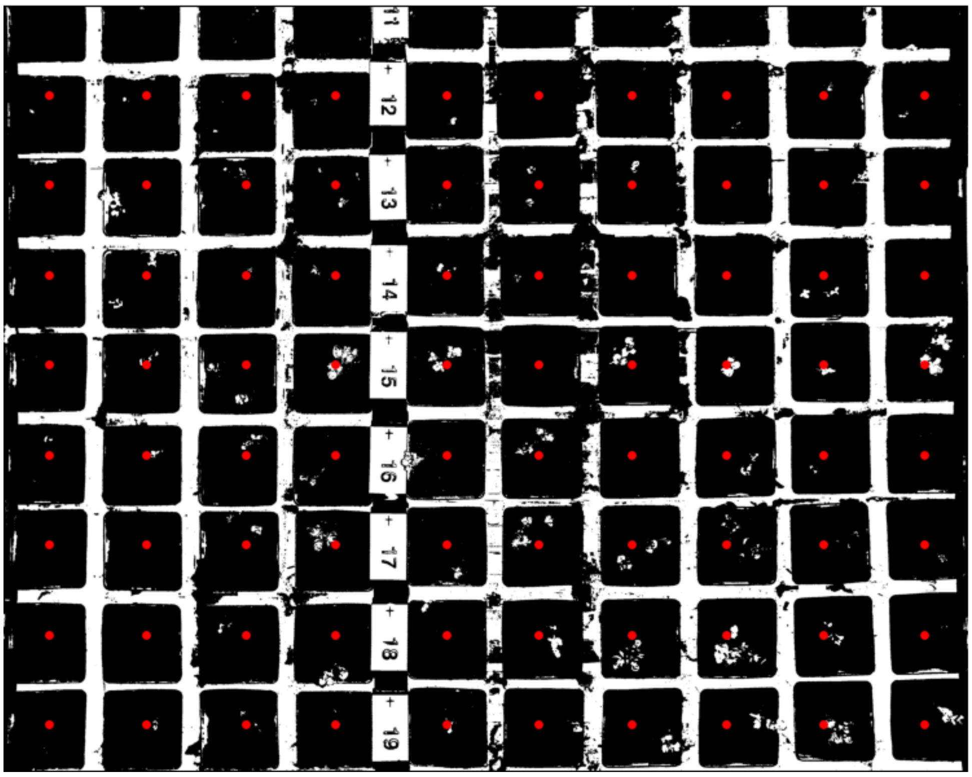


FIGURE 4. Distribution of approximate pot midpoints for the thresholding image

To find the initial centre points, it will be assumed that the number of rows and number of columns of pots on the image is known. The number of columns is ten in this case. However, the number of rows varies between eight and nine. To place the initial points in the middle of the pots, the height of the image and width of the image in pixels gets divided by the number of pots. The midpoints are then distributed over this distance in a way such that the points are spread out evenly. As often only half a pot is visible in the vertical direction, the height of the image gets corrected by where the first pot starts. The place where the first whole pot starts is calculated in a similar fashion as the before described method to find the boundary. Figure 4 gives an example of the output of this method. If there are n by k pots on a picture, the center of a pot can be described by the following simple formula:

$$\text{Midpoint}_{i,j} = (ih/k + h/(2k), \quad jv/n + v/(2n) + y_{start} \quad),$$

for $i = 0, ...k-1$, $j = 0, ..., n-1$. Here h and v are the horizontal and vertical length of the image respectively.

2.4. Pot labelling.

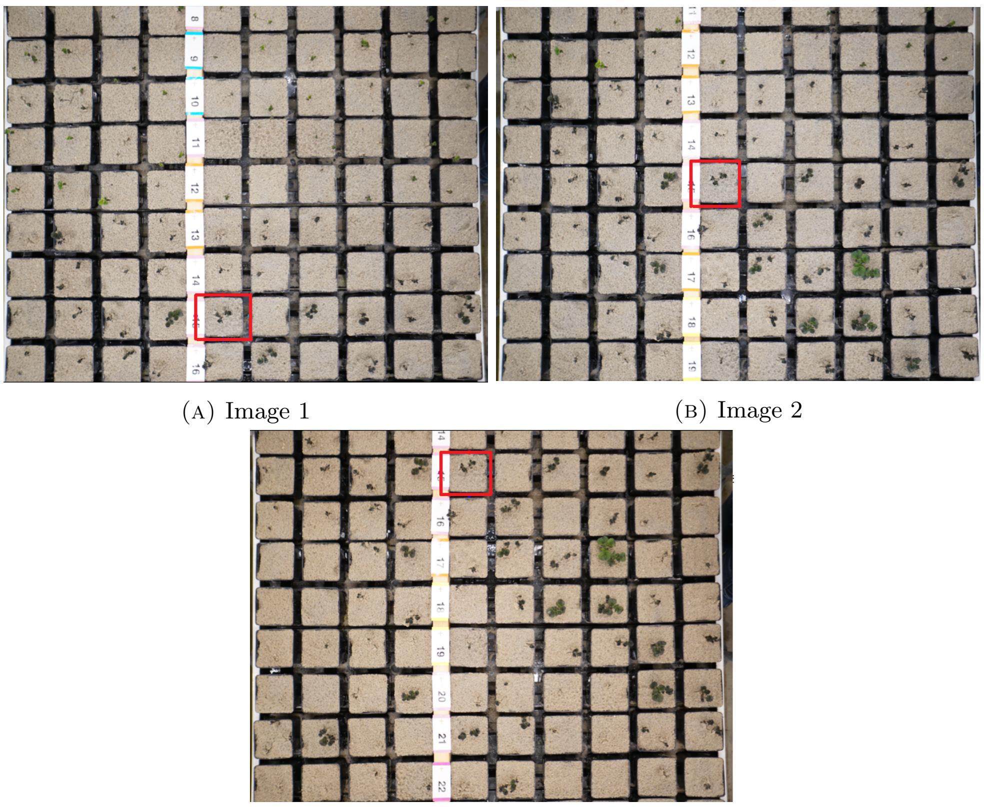
On each table, every pot has an absolute, table related, row number and a column number. From every table there are 27 images taken. In the image itself a pot is identified with the column and row number specific to the image. The pots in the images need to be labelled correctly so that they match the labelling on the table. To achieve this, a table with image names and row numbers is created. The table shows per image which (absolute) rows of pots are visible in this image. Because every pot can be found in three images (in different local rows inside the images), these three images have to be grouped together. For example, to find a pot with the absolute row number 15 and the column number 4 of table 1, the images where the pot is present have to be retrieved from the table, see Table1). Obviously, this pot appears in the images 5, 6 and 7, and it can be calculated in which local image-specific row the pot can be found. For image 5 the first absolute row of pots is row 8. Thus, the absolute row 15 is the local row $15 - 8 + 1 = 8$ in the image 5. This can then be repeated for the other images.

2.5. Pot sorting.

Three different potato varieties each divided into 30 batches, are growing on the tables. Potato plants of the same batch are placed in groups of 4 adjacent pots at random but known locations on the table. With the data set, a schematic map of these locations is provided, which contains the information on the absolute row and column of the corresponding pot. Every batch is featured by two four-pot clusters on a table containing 8 individual pots. In the schematic map the four pot groups can be found together with their corresponding absolute row and column numbers. However, since every

image	from row	to row
⋮		
Image 5	8	16
Image 6	11	19
Image 7	14	22
⋮		

TABLE 1. Example of the row-range table



(A) Image 1 (B) Image 2

(C) Image 3

FIGURE 5. Identification of the same pot in three images

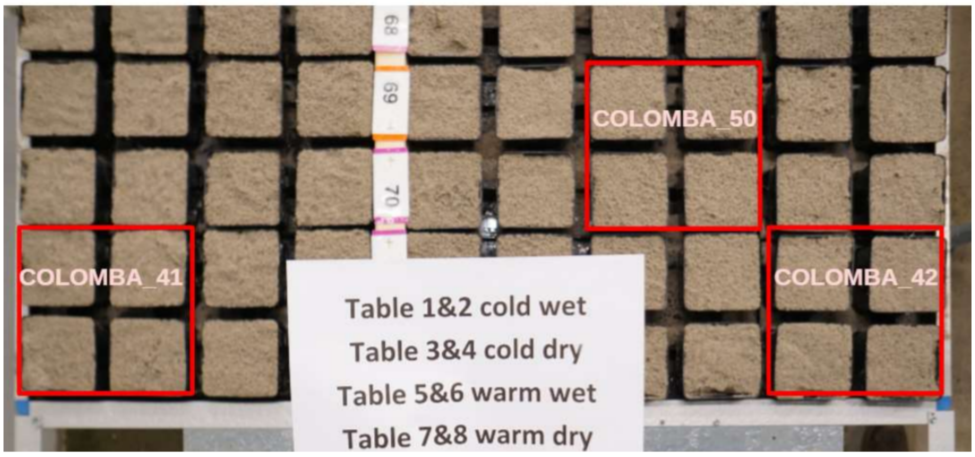
group consists of 4 pots, the group row and column locations must be converted into the corresponding pot columns and rows. Given the row n and column m of a group as given in the schematic map, the row-column locations of corresponding pots can be calculated:

$$\begin{aligned}
\text{pot } 1 &= ((n-1)\cdot 2+1, (m-1)\cdot 2+1)\\
\text{pot } 2 &= ((n-1)\cdot 2+1, (m-1)\cdot 2+2)\\
\text{pot } 3 &= ((n-1)\cdot 2+2, (m-1)\cdot 2+1)\\
\text{pot } 4 &= ((n-1)\cdot 2+2, (m-1)\cdot 2+2),
\end{aligned}$$

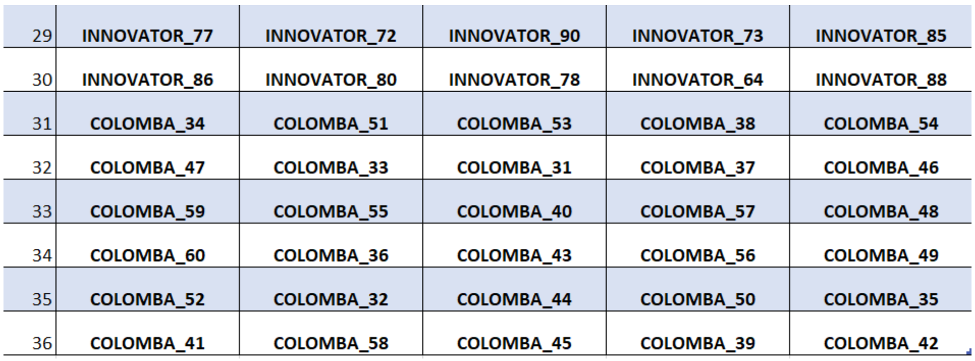
where (i, j) stands for (row, column). For example, Colomba 50 can be found in the map at row 35 in column 4. The pots corresponding to that group of Colomba 50 are:

$$\begin{aligned}
\text{pot } 1 &= (((35-1)\cdot 2+1, (4-1)\cdot 2+1)) = (69, 7)\\
\text{pot } 2 &= (((35-1)\cdot 2+1, (4-1)\cdot 2+2)) = (69, 8)\\
\text{pot } 3 &= (((35-1)\cdot 2+2, (4-1)\cdot 2+1)) = (70, 7)\\
\text{pot } 4 &= (((35-1)\cdot 2+2, (4-1)\cdot 2+2)) = (70, 8).
\end{aligned}$$

The image 6a and the map 6b illustrate this procedure.



(A) Four-pot groups on the table



29	INNOVATOR_77	INNOVATOR_72	INNOVATOR_90	INNOVATOR_73	INNOVATOR_85
30	INNOVATOR_86	INNOVATOR_80	INNOVATOR_78	INNOVATOR_64	INNOVATOR_88
31	COLOMBA_34	COLOMBA_51	COLOMBA_53	COLOMBA_38	COLOMBA_54
32	COLOMBA_47	COLOMBA_33	COLOMBA_31	COLOMBA_37	COLOMBA_46
33	COLOMBA_59	COLOMBA_55	COLOMBA_40	COLOMBA_57	COLOMBA_48
34	COLOMBA_60	COLOMBA_36	COLOMBA_43	COLOMBA_56	COLOMBA_49
35	COLOMBA_52	COLOMBA_32	COLOMBA_44	COLOMBA_50	COLOMBA_35
36	COLOMBA_41	COLOMBA_58	COLOMBA_45	COLOMBA_39	COLOMBA_42

(B) Schematic map of four-pot group locations

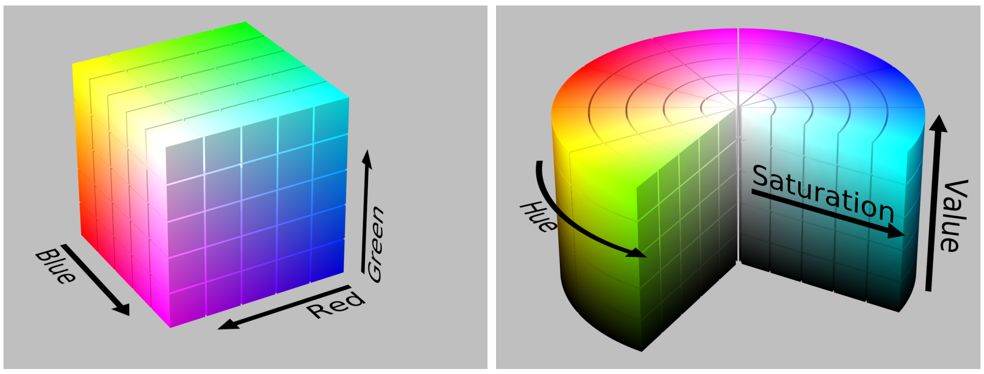
FIGURE 6. Distribution of batches over tables.

2.6. **Parallel multi-core implementation.** Modern computers have an integrated multi-core processor with around 4 till 8 CPU's (central processing units) built-in. Advanced programming languages like Java and C++ are automatically optimised to send different threads to different processors. However, due to a relatively large memory consumption and for other technical reasons the Python language is equipped with the so-called Python Global Interpreter Loc (GIL), see R. Gadde (2018) [3]. All different threads are sent to the same CPU to prevent python from crashing in case too much RAM is used. The program that pre-processes the data only uses about 150 MB, whereas modern computers these days have 4, 8 or even 16 GB RAM available. This means that the execution can be safely accelerated up by using multiple cores, which is very convenient considering the large input data set.

To bypass the GIL and still use multiple cores, Python provides packages such as `multiprocessing`. This allows closed loops to be executed parallel to each other. Closed loops are loops in the program that don't need information from other loops to run. The data pre-processing program consists of many closed loops. The program is so designed that the processing of the images of a table on a given day only uses the information of that day. Thus these Loops can be executed at the same time. Python's `multiprocessing.pool` enables it to send different closed `for` loops to different CPU's.

For one table 27 images are taken per day and each table is photographed on 11 days. Each individual image has the resolution of 5169×3448 pixels. If the data for one table must be gathered 27×11 297 photos must be processed. Per photo $5169 \times 3448 = 17,822,712$ pixels have to be processed. Hence, per table about $5.3 \cdot 10^9$ pixels are to be evaluated. With the most time-consuming operation in the program being the image thresholding. Where every single pixel has to be compared with the threshold values.

On the particular computer, where the data-preprocessing program was executed, processing the data for one table with only one core took about 11.78 minutes. If all eight tables had to be processed with one core it would have taken about 98 minutes. Executing the program for the same table with six CPU's took 2.8 minutes. This is almost six times faster, as was expected. Now, running the program for all eight tables with six cores takes about 23 minutes.



(A) RGB cube (B) HSV cylinder

Figure 7. RGB cube and HSV cylinder.
Images taken from Wikipedia [5]

3. Leaf canopy area

3.1. **Green detection.** An important aspect of the growth of a plant is the leaf canopy. To obtain the leaf canopy from the image a green detector based on HSV-filtering will be applied. The green detection is based on the detector described in the article Greenness identification based on HSV decision tree by Yang et al. [4].

An important aspect of this method is that the pixel values are converted from RGB-values to the HSV color space. In the RGB-color space a color has a Red, Green and Blue value. These values are integers between 0 and 255. Were the values stand for how bright the red, green or blue is mixed in the color. The true color of a pixel depends on the ratio of the RGB-values. Resulting in the fact that same color with different brightness have different RGB-values. The benefit of the HSV-color space is that the color of a pixel is mostly determined by the H value. HSV stands for Hue, Saturation, Value. The Hue is the color's angle. The different colors appear in the following range: red starts from 0 degrees and flows over to green, located around 120 degrees. Then green turns in to blue located at 240 degrees. And blue turns back into red at 360 degrees. The Saturation determines the colors brightness and the Value determines the mix with black or white. For a visualisation of the difference between the RGB-color space and the HSV-color space please refer to Figure 7.

For filtering the green out of the image proceed as follows: convert the RGB-values to the HSV color space. Then threshold on the H-values of the pixels such that the green color remains.

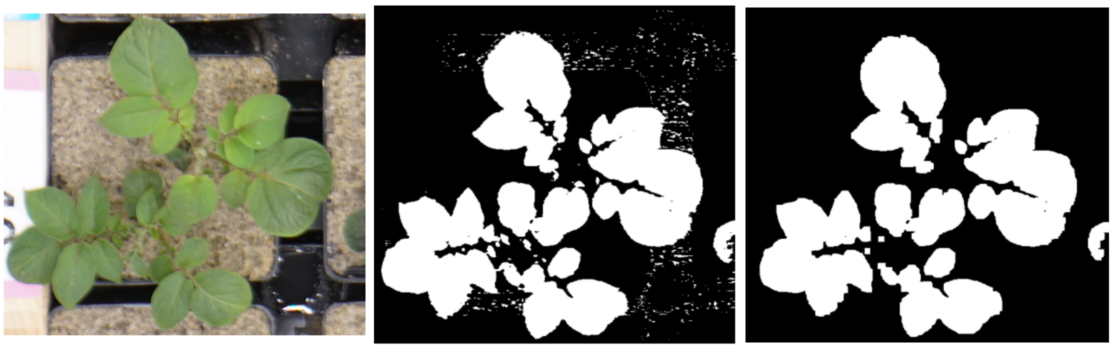
$$I(u,v) = \begin{cases} 1 & 50 < H(u,v) < 150 \\ 0 & \text{else} \end{cases}$$

In the article greenness identification [4], thresholding with boundary values of 50 en 150 degrees was used as primary filter. A secondary thresholding was used to filter out straws from the image.

$$I(u,v) = \begin{cases} 0 & 49 < H(u,v) < 60 \text{ or } 5 < S(u,v) < 50 \text{ or } V(u,v) > 150 \\ 1 & \text{else} \end{cases}$$

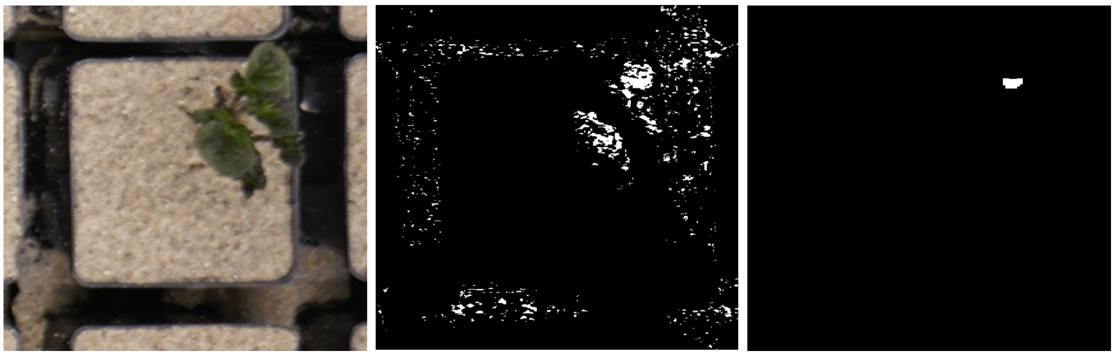
Where the H, S and V are the HSV-values of a pixel (u, v). However, in the data set of Flight to Vitality there is no straw in the background. Therefore, this step is less relevant. After filtering, the noise gets cancelled by deleting every object that is smaller than 9 pixels. For an example of the result of this HSV based filter please refer to image 8.

While testing the filter on the images, the filter did not respond to darker green leaves well. Therefore some adjustments where made to lower the lower threshold bound to 30. This too results in more noise since the color composition of the dark green leave lies very close to the colors of the darker background of the boundary between the pots. Refer to figure 9 for an example of the HSV-based filter not responding well to a dark green leave. And refer to figure10 as well.



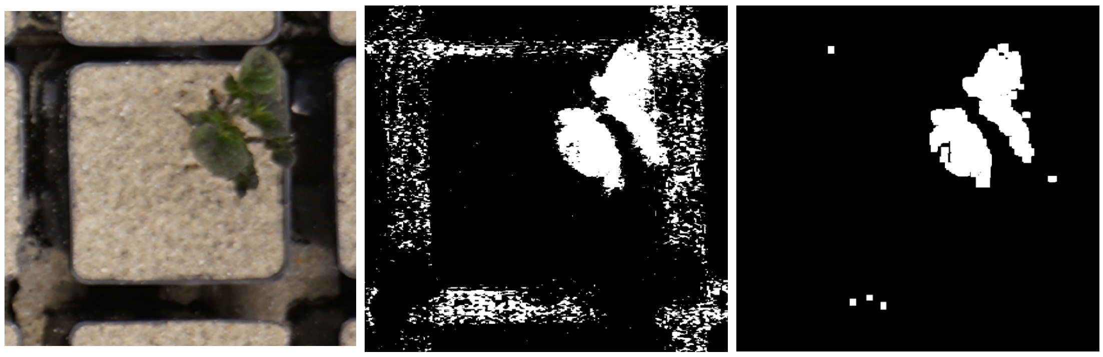
(A) Original (B) Hue threshold filter (C) Noise filter

Figure 8. HSV-based green detection



(A) Original (B) HSV threshold filter (C) Noise filter

Figure 9. HSV-based green filter applied to dark-green leaves



(A) Original (B) Tuned HSV filter (C) Tuned HSV after noise filter

Figure 10. Adaptive HSV-based green filter applied to dark-green leaves

3.2. **Canopy growth model.** When analyzing the growth of the plants, it is interesting to look at the growth of the whole population rather than to look at individual plants. The plants are in the early stage of their growth and every plant is based in its own pot. Therefore, we assume a growth model of biologically unlimited growth. In this section an exponential growth model is discussed and analyzed following [6].

Assume that the individual plants in the population grow exponentially:

$$\frac{dn_i}{dt} = \alpha_i n_i, \qquad n_i(0) > 0, \qquad i = 1, \dots P \tag{1}$$

Here α_i are the growth constants and n_i is the size of the i'th individual plant. In this project n_i will represent the plant canopy or the plant height. Instead of solving each ODE separately we look at the population distribution function $u(n, \alpha, t)$, which represents the distribution of plants over their size n and growth rate α as a function of time. Assume that the total number of plants is conserved over time. Hence,

$$\int_0^\infty \int_0^\infty u(n, \alpha, t) dn d\alpha = P.$$

Where P is the total number of plants. Integrating only over the growth rate α now gives a approximation for the distribution function of the plant size n:

$$\int_0^\infty u(n, \alpha, t) d\alpha = \rho(n, t)$$

Since the total number of plants P is conserved the distribution function u obeys a local conservation law (continuity equation) of the type:

$$\frac{\partial u}{\partial t} + \nabla \cdot \mathbf{J} = 0, \tag{2}$$

where $\mathbf{J}$ is the phase-space current density and the $\nabla \cdot$ operator differentiates with respect to the variables n and α. The phase-space current density is a vector-valued function of the following form: $\mathbf{J} = u\langle v_n, v_\alpha \rangle$, where v_n and v_α are the phase-space velocities, i.e., $v_n = \frac{dn_i}{dt}$ and $v_\alpha = \frac{d\alpha_i}{dt}$. To obtain the explicit expression for the current density, both of these velocities must be written in terms of the phase space variables n and α only, which is trivial in the present case. Namely, $v_n = \alpha n$ and $v_\alpha = 0$.

The continuity equation now reduces to the following Cauchy problem:

$$\frac{\partial u}{\partial t} + \alpha n \frac{\partial u}{\partial n} + \alpha u = 0, \quad u(n, \alpha, 0) = u_0(n, \alpha)$$

Assuming that the initial distribution over sizes and growth rates is given, this equation can be solved with the method of characteristics and predicts the evolution of the distribution in time. The characteristic equations are:

$$\frac{dt}{ds} = 1, \quad \frac{dn}{ds} = \alpha n, \quad \frac{du}{ds} = -\alpha u$$

The solutions of these ODE's are:

$$t = s + c_0, \quad n = c_1 e^{\alpha s}, \quad u = c_2 e^{-\alpha s}$$

This leads to the following relations:

$$n = C_0 e^\alpha t, \quad u = C_1 e^{-\alpha t}$$

Where $C_0 = c_1 e^{c_0}$, same for $C_1 = c_2 e^{c_0}$. On the lines $ne^{-\alpha t} = c_0$ the function is constant in t. The boundary values at $t = 0$ are used to find $c_1 = u_0(n_0, \alpha)$. Because $ne^{-\alpha t}$ is constant, u can now be expressed as follows:

$$u(n, \alpha, t) = e^{-\alpha t} u_0(ne^{-\alpha t}, \alpha)$$

Assume that the initial distribution is a two-dimensional Gaussian function. The result of the time evolution of this density function and the marginal size density typically look like Figure 11. Compare the shape of the distribution function $\rho(t, n)$ with the kernel density estimates of the leaf area. For instance, in Figure 12, the blue line is set as day zero, the green line is three days later, and the red line is day seven. In Figure 12 the density shifts to the right of the graph. The same happens for the distribution function of the simulated example 11. This gives a reason to assume the applicability of the growth model considered above to the leaf canopy.

The following example shows one way to apply the growth model to the canopy data. The data is used from day 31-01-2020, 03-02-2020 and day 08-02-2020, where the first is set to day zero. All pots that had zero canopy area where left out of the data. For finding an estimate of the growth factor α,

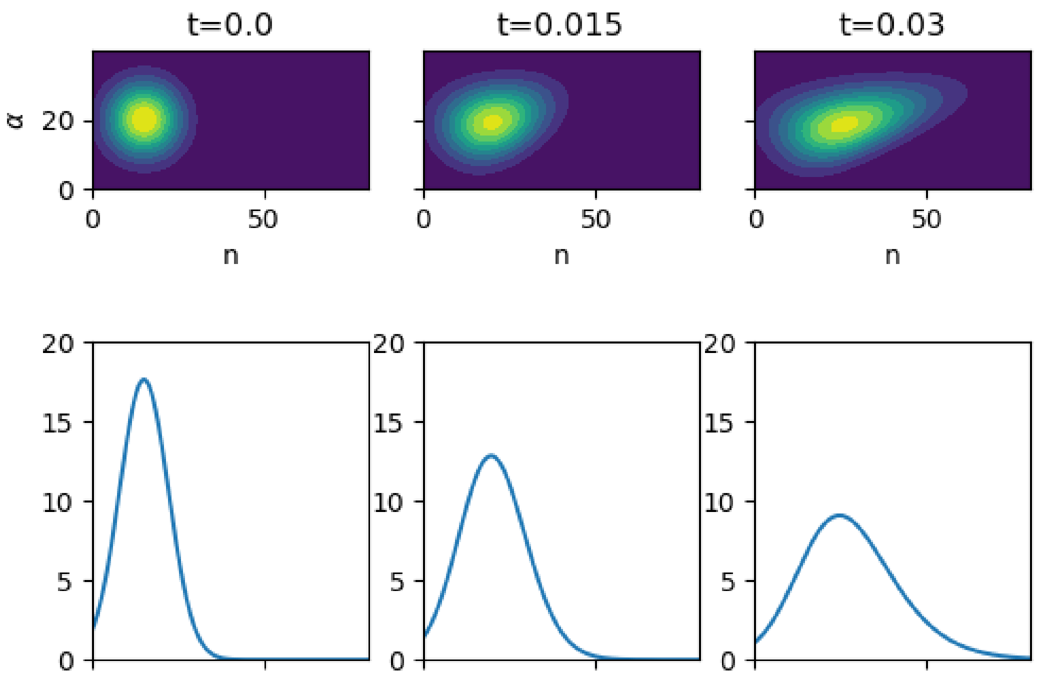


FIGURE 11. Example of an exponential growing population. Top row are snapshots of $u(n, \alpha, t)$ and bottom row is the numerical integrand $\rho(n, t)$

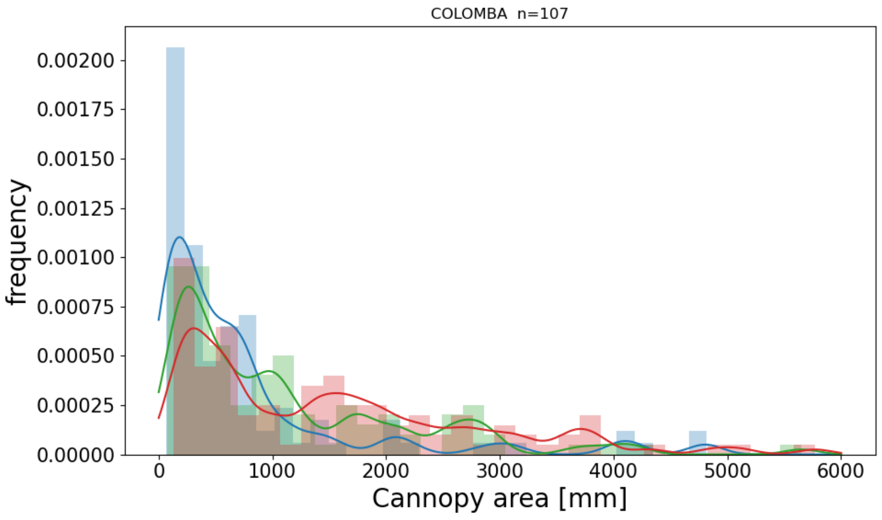


FIGURE 12. Leaf canopy of COLOMBA, table 5 6

an exponential curve was fitted to means of the canopy area data of Figure 13. As initial distribution, a Gaussian distribution is taken with the mean and the standard deviation as parameters, from the data of day 0 (day 31-01-2020). For the distribution of α a Gaussian distribution was taken as well. The results of the simulation are visible in Figures 14 and 15.

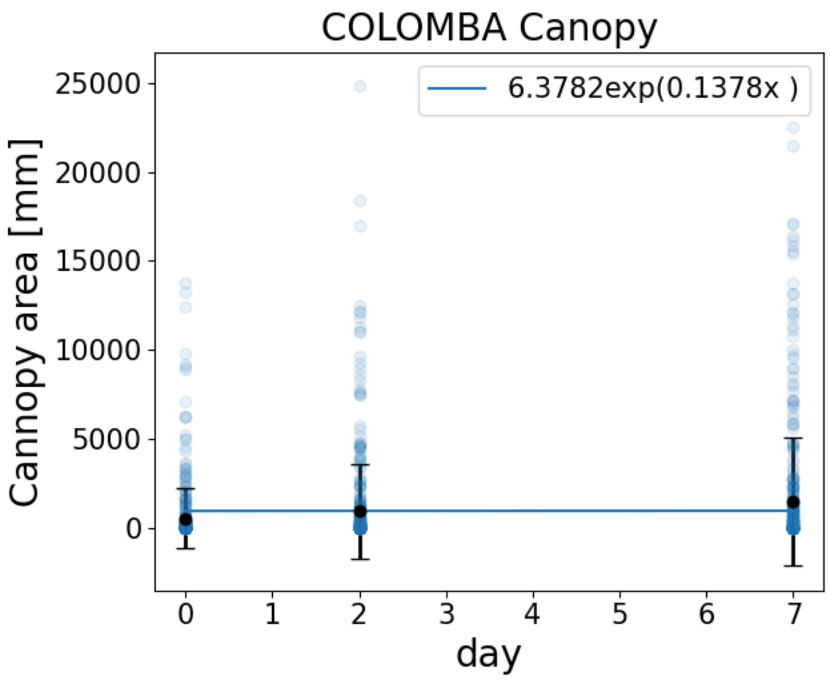


FIGURE 13. Exponential fitted curve trough days

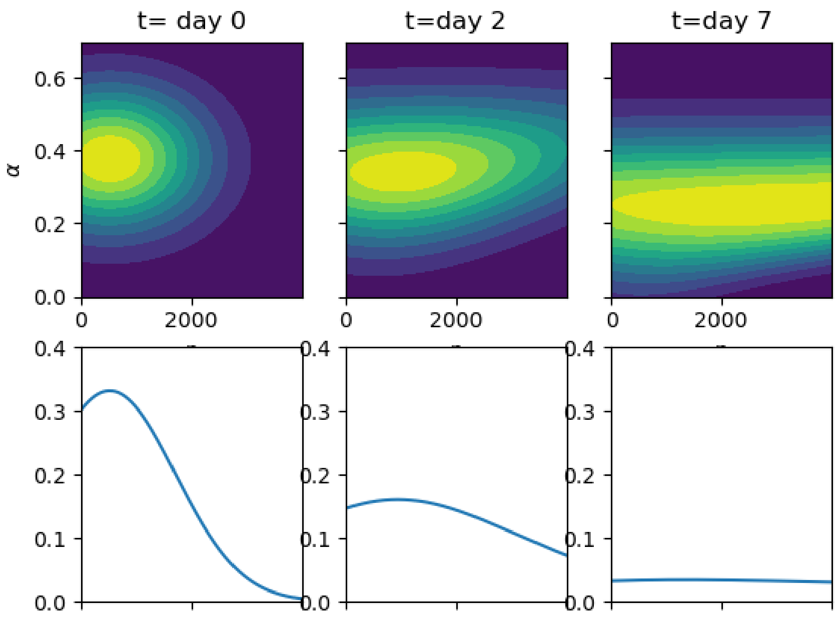


FIGURE 14. Simulation of canopy growth model $\alpha = 0.1378$

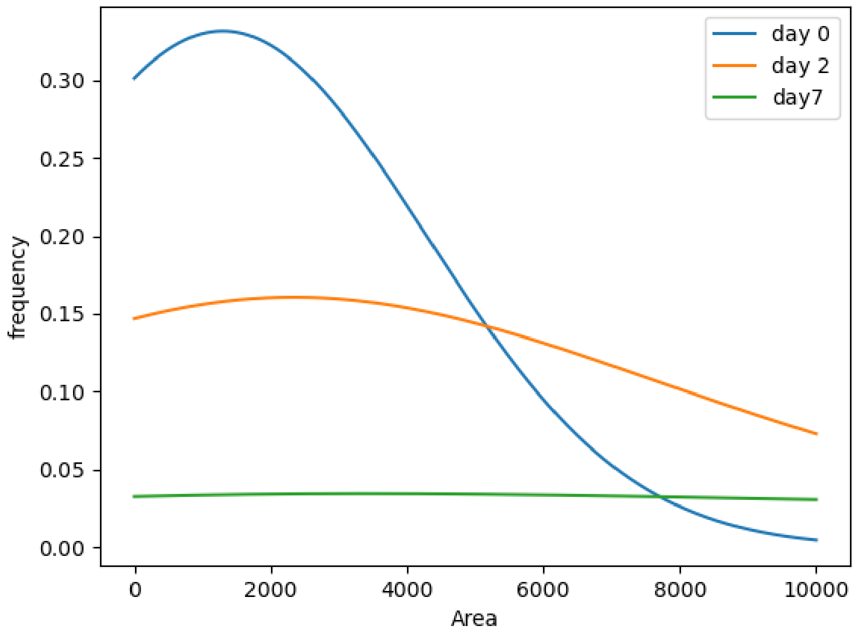


FIGURE 15. Simulation of canopy growth model $\alpha = 0.1378$. Curves plotted in one figure

Table 2. Camera parameters and imaging setup

parameter	value	units
camera model	SONY ILCE-7RM2	
sensor dimensions	36×24	mm
image resolution	5168×3448	pixels
focal length (lens), f	16	mm
height from table surface, h	~ 1095	mm
displacement along table, d	~ 380	mm
orientation	downwards	
placement	mid table	
number of tables	8	
snapshots per table	27	
container grid per table	9×10	containers

4. Plant height

The plants are photographed from three different orientations. From these perspectives and in combination with the cameras parameters it is possible to reconstruct the location of points. If two points are reconstructed, one on the pot and another on the plant, the difference of these points will return the height of the point on the plant. Here we also consider a method where the height is calculated without reconstruction of the individual points but relative to each other.

4.1. **Object point triangulation.** Let $\mathbf{r}' = [x', y', z']^T$ be the coordinates of an object point in the camera reference frame. The image of this point obtained by the camera has homogeneous coordinates $\mathbf{p} = [u, v, 1]^T$, where (u, v) are the pixel coordinates. The homogeneous coordinates are thus the pixel coordinates with one taken as z With multiple camera positions it is necessary to be careful with respect to which coordinate system the variables are defined. Take an $\mathbf{r}' = [x', y', z']^T$ in the coordinate system of the camera. This point gets projected on the camera's detector and has homogeneous coordinates $\mathbf{p} = [u, v, 1]^T$ where (u, v) are the pixel coordinates. The camera pinhole model then gives the relation between the pixels coordinates and the points coordinates.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{f}{z'} \begin{bmatrix} x' \\ y' \end{bmatrix}, \tag{3}$$

The camera's focal length $f > 0$ is assumed to be known. It is the length between the camera's sensor and the pinhole. z' is the length from the object point to the camera's pinhole. For the camera focal length f see Table 2.

The relation can also be expressed using the homogeneous pixel coordinates.

$$\mathbf{p} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{f}{z'} \begin{bmatrix} x' \\ y' \\ z'/f \end{bmatrix} = \frac{1}{z'} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \frac{1}{z'} C\mathbf{r}', \tag{4}$$

This is more convenient since the relation between the z axis are then also involved. Here, C is called the camera's matrix, it contains the camera's focal length parameter.

Using vector $\mathbf{e}_3 = [0, 0, 1]^T$, to express z as $z = \mathbf{e}_3^T r'$. And multiply both sides of equation (4) by z. Finally, move $\frac{1}{z'} C\mathbf{r}'$ to the left side of the equation. Then (4) can be expressed as:

$$\begin{aligned} &P\mathbf{r}' = \mathbf{0}, \\ &P = \mathbf{p}\mathbf{e}_3^T - C = \begin{bmatrix} -f & 0 & u \\ 0 & -f & v \\ 0 & 0 & 0 \end{bmatrix}. \end{aligned} \tag{5}$$

and

$$p\mathbf{e}_3 = \begin{bmatrix} u & 0 & 0 \\ v & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

The unknowns of this equation are x', y' and z'. The known variables are u and v. The last row of P consists of only zeros. Therefore the rank of P is maximum two. This system is solved by infinitely many nonzero solutions. It can be easily verified that the line $r' = \alpha[u, v, f]^T$, for $\alpha \in \mathbb{R}$ solves (5). What this means is that every point on a line from the cameras pinhole to the point $\mathbf{r'} = [x', y', z']^T$ gets projected in the camera on the same pixel.

Because this model uses three camera positions, the object point has three different coordinates with respect to all the different cameras. Therefore, it is better to work from the object reference frame. Then only the cameras move with respect to the point. With respect to the objects reference system let $\mathbf{r} = [x, y, z]^T$ be the coordinates of an object point. To express the point $\mathbf{r'}$ in the coordinate system of the object. The camera's coordinate system has to be translated and rotated to the system of the object point. Let R be the rotation matrix and $\mathbf{s}$ the translation vector. Then $\mathbf{r'}$ converted to the new coordinate system is as follows; $\mathbf{r'} = \mathbf{s} + R\mathbf{r}$. This in combination with (4) gives a relation for $\mathbf{r}$:

$$P(\mathbf{s} + R\mathbf{r}) = \mathbf{0}. \tag{6}$$

It is already known that $P\alpha[u, v, f]^T = 0$. So solving for r in the following equality:

$$P\alpha[u, v, f]^T = \mathbf{s} + R\mathbf{r}$$

gives the solution of (6). Namely, $\mathbf{r} = R^{-1}(\alpha[u, v, f]^T - \mathbf{s})$, $\alpha \in \mathbb{R}$. In the camera's reference frame, the pinhole of the camera is located at point $\mathbf{r'} = 0$. This point corresponds to the point on the line $(\alpha[u, v, f]^T - \mathbf{s})$ where $\alpha = 0$. The solution of (6) reduces to $\mathbf{r} = -R^{-1}\mathbf{s}$.

In the current experiment's setup, the camera positions are arranged such that every single plant is photographed three times. The camera moves over the table in the y-direction. Each camera position adds one reference frame. Because the equations are all translated to the object points reference system $\mathbf{r}$, only one coordinate system is needed instead of three. Every pixel pair gives an algebraic system of equations. Since there are three points, we get three systems of equations.

$$\begin{aligned} P_1(\mathbf{s}_1 + R_1\mathbf{r}) &= \mathbf{0}, \\ P_2(\mathbf{s}_2 + R_2\mathbf{r}) &= \mathbf{0}, \\ P_3(\mathbf{s}_3 + R_3\mathbf{r}) &= \mathbf{0}, \end{aligned} \tag{7}$$

where $P_i = \mathbf{p}_i\mathbf{e}_3^T - C$ and $\mathbf{p}_i$, $i = 1, 2, 3$, are the three image points in homogeneous coordinates. Each camera position has roughly the same orientation (downwards). And the camera's locations s_i relative to each other are considered to be known, as we know the displacement d and the height h. There are now three unknown variables $\mathbf{r} = [x, y, z]^T$ and 6 known variables $_i = [u_i, v_i, 1]^T$. Therefore, the system is over determined. Still, it is solvable:

$$\begin{bmatrix} P_1R_1 \\ P_2R_2 \\ P_3R_3 \end{bmatrix} \mathbf{r} = \begin{bmatrix} -P_1\mathbf{s}_1 \\ -P_2\mathbf{s}_2 \\ -P_3\mathbf{s}_3 \end{bmatrix}. \tag{8}$$

The reference frame of the object point can be chosen with respect to the middle camera position. Then the middle camera is assumed to be in the middle of the reference frame. The first camera is located with $-d$ in y directions from the middle camera. And the camera direction of the third camera position is d in the y direction. The translation vectors become:

$$\mathbf{s}_1 = \begin{bmatrix} 0 \\ -d \\ h \end{bmatrix}, \quad \mathbf{s}_2 = \begin{bmatrix} 0 \\ 0 \\ h \end{bmatrix}, \quad \mathbf{s}_3 = \begin{bmatrix} 0 \\ d \\ h \end{bmatrix}, \tag{9}$$

In this case h is the height of camera above the table where the plants are positioned on. The displacement d is the movement of the camera along the table in the y-direction. See Table 2. The general rotation matrices are given by:

$$\begin{aligned} R_x(\theta) &= \begin{bmatrix} 1 & & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}, \\ R_y(\phi) &= \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix}, \\ R_z(\omega) &= \begin{bmatrix} \cos(\omega) & -\sin(\omega) & 0 \\ \sin(\omega) & \cos(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \end{aligned} \tag{10}$$

Which represents the rotation of a point around the corresponding axis. In this situation the camera is faced downwards. The angle that the camera makes with the x-axis as $= \pi$. The camera is not rotated around the y, or z axis resulting $\phi = 0$ and $\omega = 0$. And in the three camera locations the camera orientation is the same. Therefore the rotation matrices are:

$$R_1 = R_2 = R_3 = R = R_x(\pi)R_y(0)R_z(0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}. \tag{11}$$

The system (8) may now be written as:

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} R\mathbf{r} = \begin{bmatrix} -P_1\mathbf{s}_1 \\ -P_2\mathbf{s}_2 \\ -P_3\mathbf{s}_3 \end{bmatrix}. \tag{12}$$

Because the system is over determined, the normal equations can be used to find the best approximation for the solution. This solution is also the solution in terms of least square error. For obtaining the normal equations the system is multiplied by its transpose:

$$R^T \left(\sum_{i=1}^{3} P_i^T P_i \right) R\mathbf{r} = -R^T \sum_{i=1}^{3} P_i^T P_i \mathbf{s}_i, \tag{13}$$

The system has a solution if the matrix of the system is non-singular. If two points or more are different then the matrix is non-singular and the system has a solution. In this case the object's point can be reconstructed by solving (13). The middle part of the system matrix is explicitly given by:

$$\sum_i P_i^T P_i = \begin{bmatrix} 3f^2 & 0 & -f\sum_i u_i \\ 0 & 3f^2 & -f\sum_i v_i \\ -f\sum_i u_i & -f\sum_i v_i & \sum_i(u_i^2 + v_i^2) \end{bmatrix}, \tag{14}$$

and its inverse is:

$$\left(\sum_{i=1}^{3} P_i^T P_i \right)^{-1} = \begin{bmatrix} \frac{1}{3f^2} + \frac{(\sum_i u_i)^2}{9f^2\Xi} & \frac{\sum_i v_i \sum_i u_i}{9f^2\Xi} & \frac{\sum_i u_i}{3f\Xi} \\ \frac{\sum_i v_i \sum_i u_i}{9f^2\Xi} & \frac{1}{3f^2} + \frac{(\sum_i v_i)^2}{9f^2\Xi} & \frac{\sum_i v_i}{3f\Xi} \\ \frac{\sum_i u_i}{3f\Xi} & \frac{\sum_i v_i}{3f\Xi} & \frac{1}{\Xi} \end{bmatrix}, \tag{15}$$

where $\Xi = \sum_i(u_i^2 + v_i^2) - \frac{(\sum_i u_i)^2}{3} - \frac{(\sum_i v_i)^2}{3}$.

4.2. **Simplified relative height formula.** Take two points with object coordinates: $\mathbf{r}_1 = [x_1, y_1, z_1]^T$ and $\mathbf{r}_2 = [x_2, y_2, z_2]^T$ that are visible in the images of two cameras. The points have homogeneous image coordinates: $\mathbf{p}_1^A = [u_1^A, v_1^A, 1]^T$, $\mathbf{p}_2^A = [u_2^A, v_2^A, 1]^T$ in image A and Consider two points with the object coordinates $\mathbf{r}_1 = [x_1, y_1, z_1]^T$ and $\mathbf{p}_1^B = [u_1^B, v_1^B, 1]^T$, $\mathbf{p}_2^B = [u_2^B, v_2^B, 1]^T$ in the image B. Images are often stored on a computer with the axis origin in the left top of the picture. Here we take the coordinate system with the origin in the image center. The pixel coordinates will be taken in the same units as the object's points. The pixels coordinate will therefore have to be converted to mm. This can be done by multiplying with the ratio of the sensor dimension and the image resolution (see Table 2)

Since the camera moves across the table, the displacement and orientation of the camera is sensitive for errors. Here, it is assumed that R is known and given by 11. However, small errors may cause the camera's orientation matrices R^A and R^B to vary from R. The same holds for the translation vectors $\mathbf{s}^A$ and $\mathbf{s}^B$. In practice they will differ from (9).

A method is developed to calculate the relative height between the two points $\mathbf{p}_A$ and bp_B, without reconstructing the two points. The height will be the relative difference in z coordinates of the points. Consider the camera parameters given as in the previous sections, this leads to the following systems:

$$\begin{aligned} P_1^A R\mathbf{r}_1 &= -P_1^A \mathbf{s}^A, \\ P_1^B R\mathbf{r}_1 &= -P_1^B \mathbf{s}^B, \\ P_2^A R\mathbf{r}_2 &= -P_2^A \mathbf{s}^A, \\ P_2^B R\mathbf{r}_2 &= -P_2^A \mathbf{s}^B, \end{aligned} \tag{16}$$

Point	Image 1	Image 2	Image 3
Q (point on plant)	(-340,1243)	(-349,7)	(-347,-1193)
P (point on pot)	(-208,1170)	(-214,-106)	(-217,-1333)

TABLE 3. Caption

For every object point there are now two camera points. And every camera point results one system. Hence there are 4 systems of equations in total, where:

$$P_i^{A/B} = \begin{bmatrix} -f & 0 & u_i^{A/B} \\ 0 & -f & v_i^{A/B} \\ 0 & 0 & 0 \end{bmatrix}. \tag{17}$$

For obtaining the relations between the two points the difference of the equations describing the same point are taken:

$$\begin{aligned}(P_1^A - P_1^B)R\mathbf{r}_1 &= -P_1^A\mathbf{s}^A + P_1^B\mathbf{s}^B, \\ (P_2^A - P_2^B)R\mathbf{r}_2 &= -P_2^A\mathbf{s}^A + P_2^B\mathbf{s}^B,\end{aligned} \tag{18}$$

Take for $s^A = [0, 0, h]$ and for $s^B = [0, d, h]$. The system can be written as follows:

$$\begin{bmatrix} 0 & 0 & u_i^B - u_i^A \\ 0 & 0 & v_i^B - v_i^A \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} h(u_i^B - h_i^A) \\ -fd + h(v_i^B - v_i^A) \\ 0 \end{bmatrix} \qquad \text{for } i \in 1, 2$$

which further reduces to

$$\begin{aligned} -(u_1^A - u_1^B)z_1 &= -h(u_1^A - u_1^B), \\ -(v_1^A - v_1^B)z_1 &= -fd - h(v_1^A - v_1^B), \\ -(u_2^A - u_2^B)z_2 &= -h(u_2^A - u_2^B), \\ -(v_2^A - v_2^B)z_2 &= -fd - h(v_2^A - v_2^B). \end{aligned} \tag{19}$$

The camera's displacement is in the y-directions. This is parallel to the pixels v-axis. From the displacements in the x- or u-directions it is not possible to reconstruct the z-coordinates. The displacement of the camera is in the v-direction, so the u-coordinates $u_1^A \approx u_1^B$, and $u_2^A \approx u_2^B$. The equations in (19) containing $u_i^{A/B}$ will be approximately 0 and not much information can be gained from them. Nonetheless, since the image v-coordinates are parallel to the cameras motion direction the relative height can be constructed from the equations containing the v-coordinates:

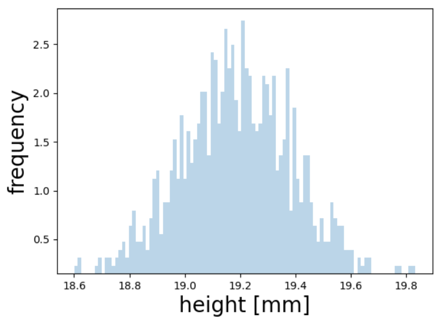
$$z_1 - z_2 = \frac{fd}{v_1^A - v_1^B} - \frac{fd}{v_2^A - v_2^B}. \tag{20}$$

4.3. **Stability of height reconstruction.** An example will be given of the height reconstruction mentioned above. And the stability will be analyzed using a Monte Carlo simulation.

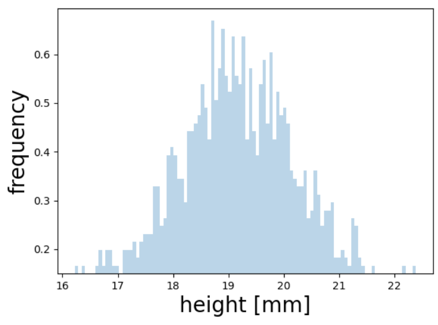
The calculated height for $d = 2 \cdot 360$ is:

$$z1 - z2 = \frac{fd}{v_1^A - v_1^B} - \frac{fd}{v_2^A - v_2^B} \approx 19.18[mm].$$

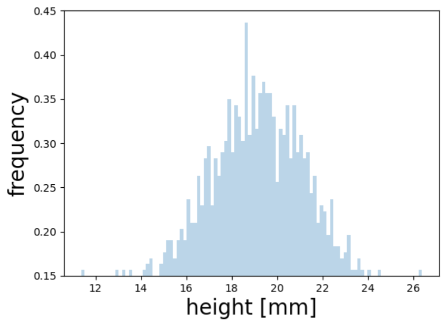
The points that are taken correspond to the first camera position and the third. The results of the Monte Carlo simulation are given in figure 16-19. The errors in the camera's parameters are taken from independent normal distributions.



(A) $\sigma = 0.01\%$

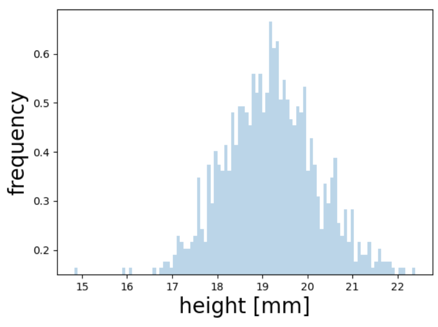


(B) $\sigma = 0.05\%$

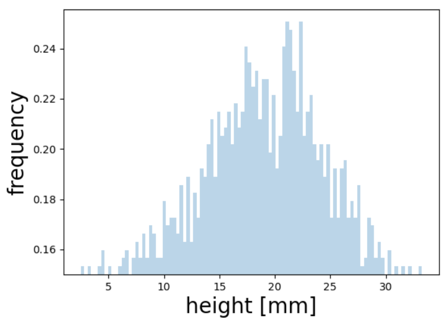


(C) $\sigma = 0.10\%$

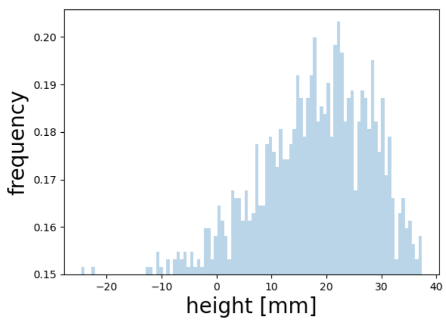
FIGURE 16. Error on displacement, true value h=19.18. Standard deviation σ is taken relative to true displacement $d = 350$mm



(A) $\sigma = 0.01\%$

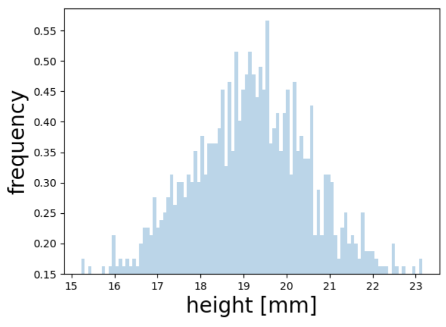


(B) $\sigma = 0.05\%$

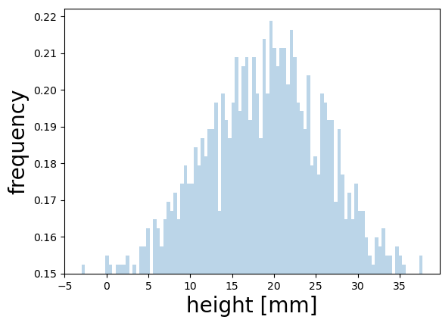


(C) $\sigma = 0.10\%$

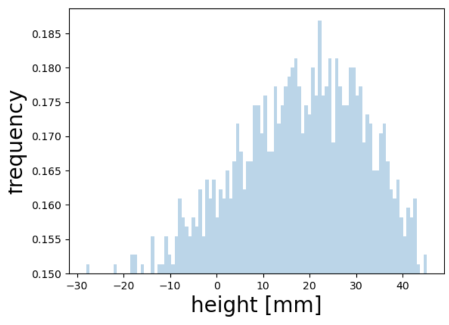
FIGURE 17. Error on θ (rotation parameter of x-axis), true value h = 19.18. Standard deviation σ is taken relative to half a circle π.



(A) σ = 0.01%

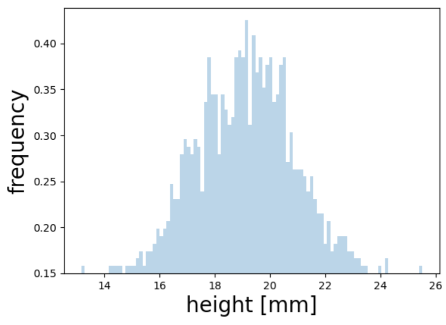


(B) σ = 0.05%

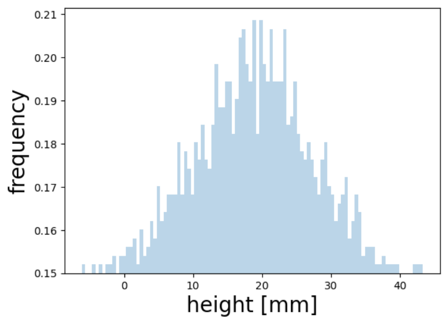


(C) σ = 0.10%

FIGURE 18. Error on ϕ (rotation parameter of y-axis), true value h = 19.18. Standard deviation σ is taken relative to half a circle π.



(A) σ = 0.01%



(B) σ = 0.05%

FIGURE 19. Error on parameters: displacement, θ, ϕ (rotation parameter of y-axis), true value h = 19.18. Standard deviation on displacement is taken relative to d = 350 mm. Standard deviation σ is taken relative to half a circle π.

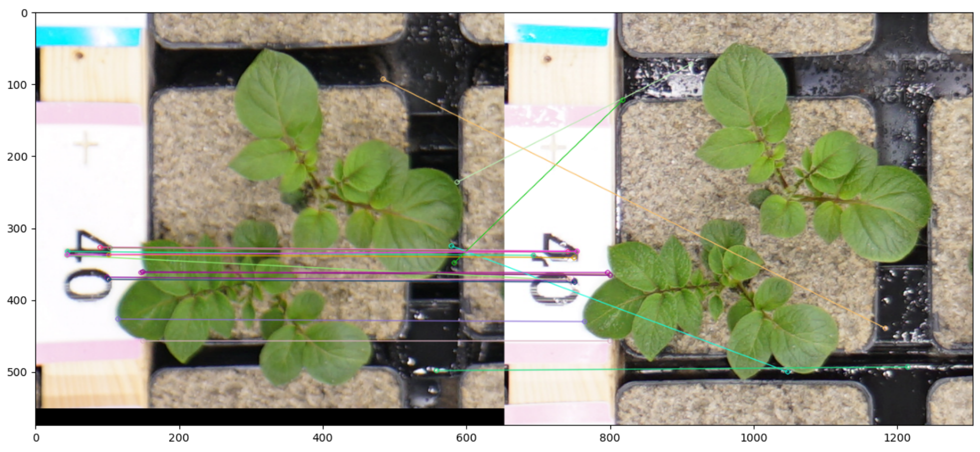


FIGURE 20. ORB Brute-Force Matcher algorithm in Python as described in OpenCV [7], hereafter called BF-Matcher.

5. AUTOMATED KEY POINT DETECTION

In this paper, the height of a plant is calculated using two points. One point that is used must be located on the pot itself. The other point must be on the plant. These key points must be present in at least two different camera images. Then, from the perspective the height can be calculated. This chapter will discuss the procedure of finding key points. Some existing methods will be discussed and a method specifically suitable for the present setup will be proposed.

One method that was applied to the images was the ORB brute force matcher 0f [7]. The BF-Matcher looks for features (key points). The feature detecting is based on finding corners. First it looks for features in one image. Then, it tries to match these features with features in the second, target image. In the images of the "Flight to Vitality" data set, the BF-Matcher did not return any points of value. Often the matcher only returned points on the pot and no points on the plant where found, see Figure 20.

To aim the BF-Matcher a bit more, it was tested in combination with the HSV-filter. Before matching, the leaves where filtered out and the background was set to black. For an example of the result see figure 21.

There are also some automated methods developed for the reconstruction of plant height. For instance in 3D Imaging of Greenhouse Plants with an Inexpensive Binocular Stereo Vision System by Li et al. [8] a stereo matcher is applied, comparing images pixel by pixel (ad-census adaptive support weight approach). There, however, the camera displacement is very small and perfectly fixed, compared to the present set up. With the larger and unstable camera displacement, the pairs of images that have to be compared are not similar enough to each other to apply a pixel by pixel comparison technique like ad-census.

For the images used in this paper a key point detector was developed based on the HSV-filter described in Chapter 3: Green detection and the image thresholding. The detector is quite simple but effective. In the Section 2.3 Cutting out of induvidual pots, a method was established that could find the boundary of the pots. The point on the pot that is taken is the middle of the pot, calculated with the values gained from the estimated pot boundaries. For the point on the pot, the plant is filtered first by the HSV-filter. Then the upper, lower, right and left point of the plant are taken (when looking from above to the plant). With these points the four relative heights are reconstructed and the average of these four heights is taken as the plant's height. When the Green identification works properly, this results in a rough, but robust estimate of the height of the plant.

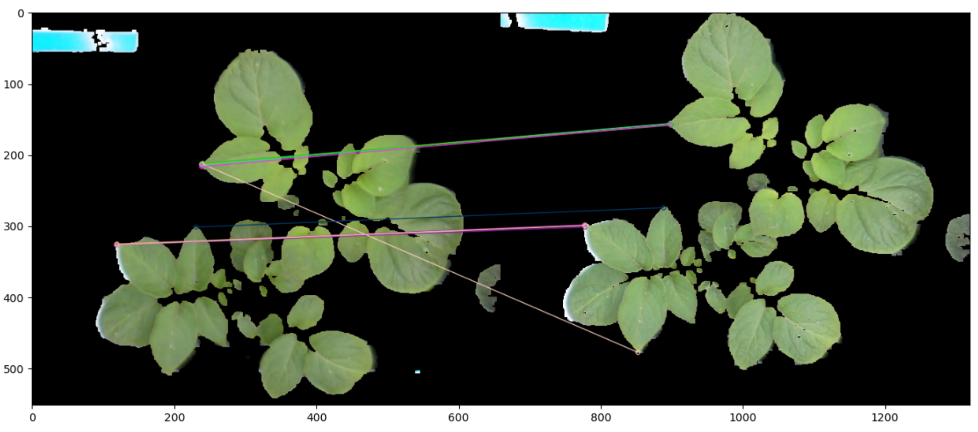
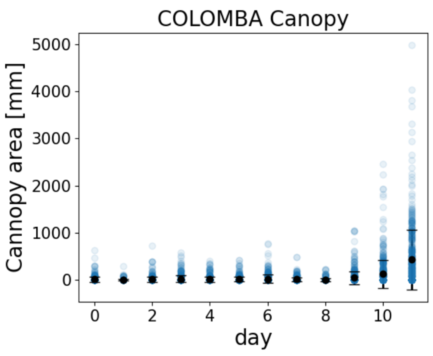
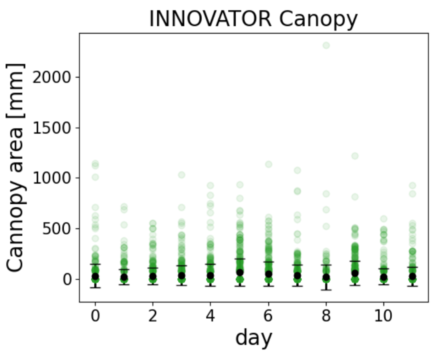


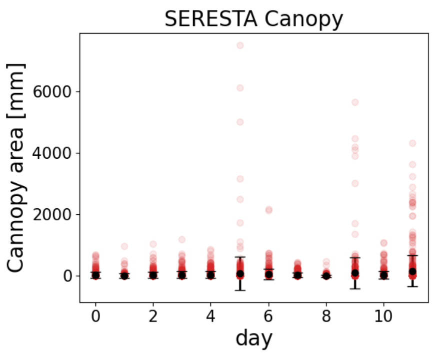
Figure 21. BF-Matcher in combination with HSV-green detection



(A) Colomba canopy



(B) Innovater Canopy



(C) Seresta

FIGURE 22. Box plot canopy table 3 and 4

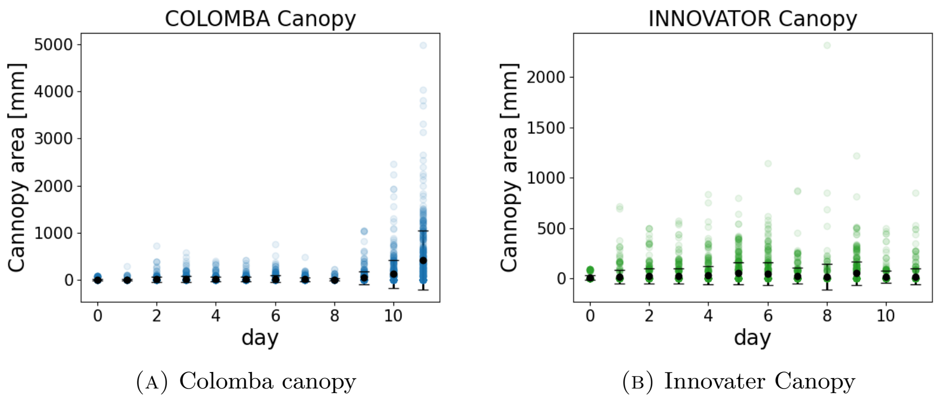
6. RESULTS

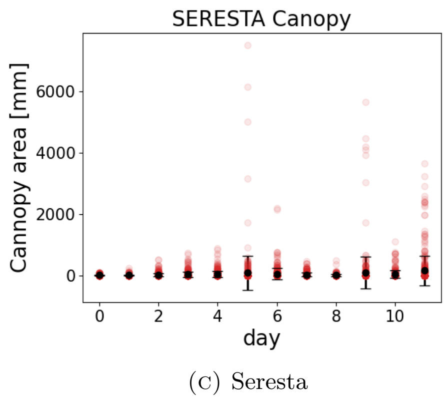
6.1. **Leaf canopy area.** The canopy of the leaves is calculated and plotted in box-plots for visually of growth rate. The canopy area was measured in square pixels and then roughly converted to mm^2. The factor which the pixel canopy is multiplied with is:

$$factor = \left(\frac{\text{sensor dimension}}{\text{image resolution}} \cdot \frac{h}{f}\right)^2 = \left(\frac{36[\text{ mm}]}{5168[\text{ pixel}]} \cdot \frac{1095[\text{mm}]}{16[\text{mm}]}\right)^2 \approx 0.2273 \frac{\text{mm}^2}{\text{pixel}^2}$$

Sometimes however, the canopy filter appears to be a bit noisy. In figure 25b at day zero the leaf canopy of the Innovator plant has already some high results. This is not possible since the plant has not even grown yet. Therefore bad results are filtered out by scratching the plants that have a high leaf area in day zero (see image 23).

See the Appendix for all the box plots of the canopy. The kernel densities are plotted. In order to see how the distribution of the leaf sizes evolve in time. See figure (24). Plants with zero leaf areas are excluded in these plots. The rest of the kernel density plots can be found in the appendix. Of table 7 and 8 no kernel densities are plotted since these results were not good enough for good plots.





(A) Colomba canopy (B) Innovater Canopy (C) Seresta

Figure 23. Box plot canopy table 3 and 4 filtered results

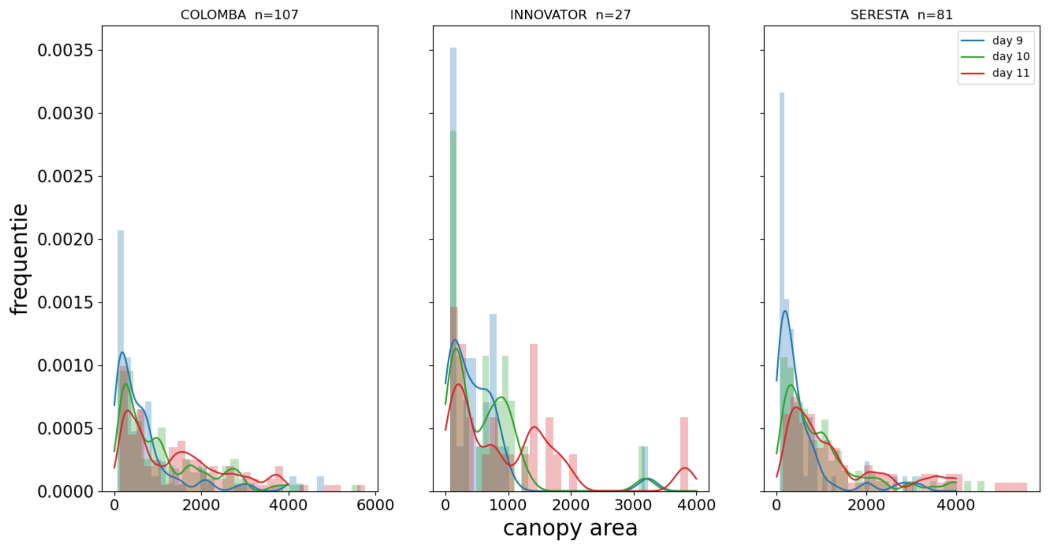
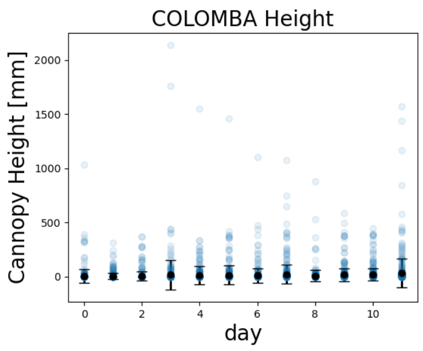
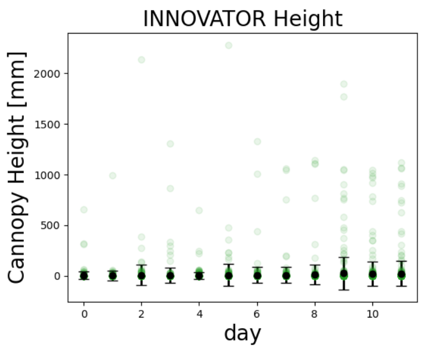


Figure 24. Kernel density of leaf canopy table 5 and 6

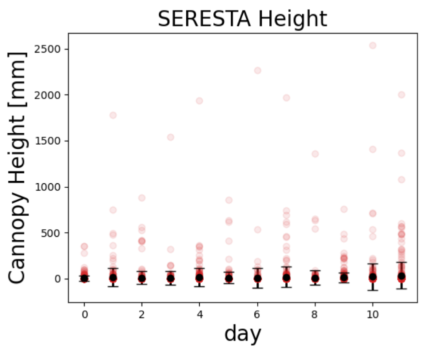
6.2. **Plant height.** In figure 25 an example of constructed height results is given. The results appear random and are therefore not very meaning full. That is why the other height plots are left out of this paper.



(A) Colomba canopy



(B) Innovater Canopy



(C) Seresta

FIGURE 25. Box plot results plant height table 1 and 2

7. Conclusions Discussion

The data pre-processing algorithm proposed in this Thesis and its Python implementation seem to work as expected and successfully extract the images of individual pots from overlapping photographs taken over many days, although the algorithm could still use some more testing on exceptional cases. For instance, on the last image of each table the program sometimes has a hard time cutting out the pots. The reason is that with the algorithmic ‘Distribution’ of the middle points of the pots, it is assumed that there are from 8 to 9 rows of pots on each image. This is not the case in the last images taken at the end of the tables. The parallel multi-core implementation of the program works very well, significantly accelerating the data pre-processing step.

The HSV-based green detector developed and tuned in this Thesis still seems to be too sensitive for noise. For instance, Figure 25b corresponding to the day zero shows that some plants already have a really high canopy area, while the plants have not really been growing yet. Nevertheless, upon some additional filtering, in most cases, the canopy area seems to have growth patterns as expected.

One of the reasons that the HSV-based filters produce noise is that they are sensitive to thresholding parameter of the Hue value. The adjustment of this parameter by only about five degrees can increase the amount of noise dramatically. In this paper the threshold value for the Hue-parameter was determined beforehand and remains the same for every table and for every day. Since the different days and different tables had other luminescence conditions keeping the Hue threshold the same for all those images does not seem optimal. By inspecting some of the HSV-filtered images, it became clear that the noise appeared as many disjoint pieces that where smaller than the plant. On the assumption that the noise is smaller than the plant, a way to improve the HSV-based green detector is to first filter the images with a rough Hue threshold value (the way it is implemented now). Then, look at the connected components and try to get a user feedback for a better Hue threshold value out of the image. Filter the connected components on their size and use the Hue-values of the smaller connected components to threshold the image again. The number of connected components can also be used as an indicator of the success of the filter. Many connected components means a noisy filtered image. While when the image is filtered well, only a few connected components will be present.

The results of the height reconstruction of the plants appear to be very random, (see Figure 25). The method that was obtained to find the key points that are used to reconstruct the heights of the plant also depends on the HSV-filter. Not optimal functioning of the HSV-filter effects the chosen key-points and therefore influences the reconstructed height. In Chapter 4.3 it is shown that small errors in the camera’s orientation parameters lead to different outcomes in the reconstructed height as well.

In the kernel density plots, (see Figure 24), a shift of mass to the right over time is visible, as the plants are growing as expected. Comparing the kernel densities with the modelled density functions in Chapter 3.2 some similarities are visible. Hence, the plant growth can be modelled with this population growth model. With this model a method can be obtained to find the plant populations growth parameter α and its distribution.

References

[1] E. R. Davies, *Computer Vision: Principles, Algorithms, Applications, Learning*. Academic Press, 5 ed., 2017.

[2] "Color to grayscale." `https://en.wikipedia.org/wiki/Grayscale#Converting_color_to_grayscale`. accesed: 2020-07-22.

[3] R. Gadde, "Using multiprocessing to make python code faster." `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm`, 2018. accesed: 2020-06-14.

[4] W. Yang, S. Wang, X. Zhao, J. Zhang, and J. Feng, "Greenness identification based on hsv decision tree," *Information Processing in Agriculture*, vol. 2, pp. 152–160, 2015.

[5] "Hls and hsv." `https://en.wikipedia.org/wiki/HSL_and_HSV#Saturation`. accesed: 2020-05-04.

[6] N. V. Budko and F. J. Vermolen, "Phase-space analysis of large ode systems using a low-dimensional conservation law," *arXiv:1503.00922, Dynamical Systems*, 2015.

[7] "Using multiprocessing to make python code faster." `https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html`, 2018. accesed: 2020-05-04.

[8] D. Li, L. Xu, X.-S. Tang, S. Sun, X. Cai, and P. Zhang, "3d imaging of greenhouse plants with an inexpensive binocular stereo vision system," *Remote. Sens.*, vol. 9, p. 508, 2017.