Automatic counting of Canola flowers from in-field time-lapse images

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Javier Garcia Gonzalez

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

The combination of plant phenotyping and computer techniques has gained popularity amongst breeders and computer scientists. The recent evolution of the latter has allowed High-Throughput Phenotyping (HTP) to play a significant role in filling the genotype-to-phenotype gap. While most of the related work in HTP is performed in controlled environments, such as greenhouses, that allow automatic devices to capture the data reliably, research in in-field phenotyping is not as robust due to environmental confounds (i.e., fog or sun-reflections). The usage of high temporal density data has not been exploited to the same degree as high spatial resolution information. However, many phenotypes (e.g., canola flowering) have a temporal component. In this document, we present an image-processing-based method that attempts to detect and count flowers of canola during the early flowering stage on in-field time-lapse images. This approach can be used to analyze the evolution of the flower density of canola plants over short periods of time during the first days of flowering thanks to the availability of high temporal resolution images. We used images extracted during Summer 2016 to generate ground truth, tune the flower detection method and count the flowers during the first days of the flowering period. We provide an overview and a discussion about additional steps that might be needed to overcome the impact of sunlight reflection on canola leaves in the detection of flowers.

Acknowledgements

I would like to thank my supervisor, Dr. Ian Stavness, and professors Dr. Kevin Stanley and Dr. Mark Eramian for their guidance and support throughout this program. Special thanks to all my colleagues from DISCUS lab for their support, help, collaboration and hours of work together.

To my wife, my parents and my family for their unconditional and constant support during this journey.

Contents

List of Tables

List of Figures

List of Abbreviations

AAFC	Agriculture and Agri-Food Canada
CIELab	*Commission Internationale de 'clairage* Luminosity, red-green (a), blue-yellow (b)
DoG	Difference of Gaussian
DoH	Determinant of Hessian
GUI	Graphical User Interface
HoFC	Histogram of Flower Counts
HSV	Hue-Saturation-Value
HTP	High-Throughput Phenotyping
LOF	List of Figures
LoG	Laplacian of Gaussian
LOT	List of Tables
MaEI	Maximum-Error Image
MiEFC	Minimum-Error Flower Count (Flower count of MiEI
MiEI	Minimum-Error Image
MSE	Mean-Squared Error
NDVI	Normalized Difference Vegetation Index
NIR	Near-infrared
RGB	Red-Green-Blue
SIFT	Scale-Invariant Feature Transform
SURF	Speed-Up Robust Features
YCrCb	Luminance (Y), red-difference (Cr) and blue-difference (Cb) chroma components

Chapter 1

Introduction

1.1 Motivation

With the recent developments in technology, farming and computers have been linked for a common objective: to increase and facilitate crop production. We can read about multiple studies that benefit from including imaging techniques using modern devices such as drones [24], or multi-spectral cameras [64]. Image-based methods can perform plant classification on numerous pictures of plants based on their observable characteristics and features - this process is called *image-based phenotyping*. Often these techniques use information that the human eye cannot see, such as multi-spectral images that capture light frequencies outside the visible range.

An interesting area that can benefit from this agriculture-computer combination is the study of canola. It is responsible for the production of canola oil, used for cooking (among other applications), and Canada one of the top producers of canola oil in the world. The provinces of Alberta, Manitoba, and Saskatchewan have the most significant production of canola plants within the country, yielding to a great opportunity of studying this plant. In 2017, farmers reported 22.8 million acres of canola planted in Canada (12.1% more than 2016), more than half of which were planted in Saskatchewan (12.6 million acres, 13.6% more than 2016) [4]. The Canola Council of Canada has reported on its 2016's annual report that canola has contributed $26.7 billion to the Canadian economy [1]

There exists a relationship between the canola oil produced and the number of flowers present in canola plants: the oil is extracted by crushing the seeds produced by the plants. These seeds are contained in productive pods, which in turn, are produced by the canola flowers. The Canola Council of Canada, [2], describes in detail the growth of canola plants, including their flowering stage, determining that the flowering stage can last between 14 and 21 days. By counting the flowers, we can determine the growth rate of the canola flower density, which in turn can relate to the maximum number of flowers produced, thus allowing for an estimation for the number of pods and seeds produced. Image processing is needed to automatically count flowers of canola using in-field images during the flowering stage.

Genotyping is the analysis and study of the genetic constitution of an organism, while phenotyping is defined as the analysis and study of observable physical features. Genotyping has provided larger amounts of data compared to phenotyping [63]. Even though during the last decade computer-aided techniques have been used for plant phenotyping to accelerate its processes, there are still many limitations in this field (also referred to as the *phenotyping bottleneck* [39]). Most of the current phenotyping procedures rely on low-throughput technologies: ground truth extraction techniques often require manual measurements of the physical properties of plants, which are not only time-consuming but also error-prone procedures. Examples include measuring the height of a plant with a ruler or manually counting flowers or leaves of a plant. These methods might result in unreliable measurements, which will eventually lead to inconsistent results.

Field-based phenotyping often focuses on providing high spatial resolution for image processing. Examples include ground-based platforms or multi-spectral drone imaging. However, many phenotypes have essential temporal components, such as growth stages; therefore there is a need for new imaging systems that have high temporal resolution, such as time-lapse cameras.

We define *high-throughput phenotyping* (a.k.a, HTP) as analyzing and capturing physical characteristics of plants using automated or semi-automated data acquisition and processing tools, which are capable of handling large quantities of information (e.g., classification of plants through image processing using automated cameras). HTP is most commonly performed in controlled environments, such as greenhouses or growth chambers that use automation to speed up image acquisition, thus increasing the total throughput. Within a controlled environment, the image acquisition system can be set up so that the pictures are acquired free from undesired effects (e.g., plants placed either in front of flat-coloured backgrounds to facilitate image segmentation, or on separated pots to avoid overlapping leaves). Additionally, cameras can be set up at precise angles that are tailored to the phenotypic information sought.

However, many of these approaches are not possible in the field, where plants grow close together in a canopy, backgrounds are uncontrolled, lighting is variable minute-to-minute or plants move from frame to frame due to the wind. Additional image processing steps are often required to overcome these limitations [36]. Nonetheless, controlled environments do not necessarily emulate the real conditions to which outdoor fields are exposed. Outdoor fields are often exposed to significant temperature or humidity variability absent in growth chambers.

HTP with high temporal resolution may be valuable in many phenotyping contexts for which high spatial resolution might fail to provide relevant data. Temporally-dense data can provide information of flowering-related phenotypes, e.g., onset, duration, termination of flowering, as well as the number of flowers, which are critical in plant breeding. Canola plants and flowers contain important temporal characteristics [2], [51]. It takes between one and two weeks for canola plants to reach maximum flowering. This fact requires canola flower counting to be performed with enough temporal resolution so that the flower density growth rate can be reliably measured. It is not feasible to count the canola flowers of large fields every day manually. There is a trend towards larger breeding experiments with larger crops, where manually counting flowers would be

highly time-consuming. A more reasonable approach is to have cameras frequently taking pictures of the crop and use their images to count the number of flowers.

However, having temporally-dense in-field images of canola plants during the flowering season can be challenging: variable weather conditions, such as rain, fog or direct sunlight, can affect not only the image acquisition equipment but also the pictures obtained themselves. While some of these effects require additional image processing steps, others should be anticipated and prevented *a prori* during data acquisition.

1.2 Solution

This thesis proposes a method that takes advantage of using temporally-dense images of outdoor canola fields extracted from time-lapse cameras. It provides a count of the number of canola flowers in an area of interest, which can be used to obtain a graphical representation of the growth of the number of canola flowers over time. Our method makes use of images taken by time-lapse cameras during the flowering season of canola plants placed in a waterproof casing to avoid water damage. These cameras record pictures automatically 24 hours a day, which reduces the human effort needed to acquire the in-field data.

It is unavoidable to encounter side effects in the extracted images due to harsh weather and other natural phenomena (e.g., direct sunlight hitting the canola plants, which can affect the flower detection process). While the cameras used in our experiment were not prepared to reduce these effects, affected images can be identified and removed. Such photos are expected to provide inconsistent flower counts. Having a high temporal resolution could allow us to discard these images and use only the subset of pictures not affected by weather, which are expected to provide a much more reliable and consistent flower count. However, there are some cases where we cannot use the high temporal resolution to solve this problem, and extra steps would be needed to get rid of the affected images.

1.3 Contributions

The contributions of this thesis include:

1. An image processing pipeline designed to detect canola flowers in in-field time-lapse images, which is focused and applied to images taken during the early flowering stages. This pipeline relies on the colour of the flowers. It uses both the RGB and CIELab colour spaces combined with a blob detector to identify canola flowers.

2. An analysis of different parameter combinations of the image processing pipeline and the accuracy offered by each.

3. Ground truth data acquisition through manual flower counting performed on a subset of images from early flowering days.

4. A discussion of different approaches that can be used to filter out images in which our method has detected flower count that is far off from the actual number of flowers. We offer an analysis of the data observed, summarize each approach's advantages and disadvantages and their expected results.

5. A discussion of how some of the challenges faced during the entire process have been overcome during the image pre-processing step, and a list of recommendation to improve data acquisition process.

1.3.1 Thesis organization

This thesis will be organized as follows:

- Chapter 2 provides an overview of the related work in indoors and outdoors image-based phenotyping. We will provide references to other methods that aim to detect some aspects of plants.

- In chapter 3 we give a brief description of the acquired data, such as the field description, cameras set up or data availability. Additionally, we describe the main challenges that we had to face during the image analysis and acquisition.

- Chapter 4 describes the pre-processing that we applied to the images, such as how the plot boundaries are defined in the time-lapse images or extraction of the timestamp from the pictures. These are one-time procedures that must be applied right after data has been collected.

- In chapter 5 we describe the flower detection pipeline, which is the one in charge of detecting the yellow flowers within the plot boundaries and count them. Additionally, we define the parameter tuning that we have followed and its results.

- Chapter 6 offers a discussion on methods that try to overcome a problem that we have found throughout the entire method development. We found that images taken on the same day and plot can provide very different flower counts. We discuss the cause of this problem and various methods that might be able to solve it.

- Finally, the thesis finishes with an overview of 2017's data as future work. We propose a set of steps to be taken to use 2017's season images and a brief explanation about which cameras provided images that might be more useful to improve our flower detection method

Chapter 2

Related work

In this chapter, we will discuss previous work conducted in the area of phenotyping performed indoors and outdoors using image processing approaches. We will present a variety of papers which describe different methods that are related to this thesis. This chapter describes how previous researchers have developed their algorithms and achieved the desired results. Previous publications have proved to be very helpful and have significantly contributed to research in image-based phenotyping. However, most of the related work focuses on indoors imaging as the data source. Outdoor imaging requires additional processing to overcome natural phenomena (as mentioned in Chapter 1). Similarly, we have also found that phenotyping using temporally-dense data (e.g., time-lapse imaging or evolution of plants over time) is not as popular as spatially-dense data (such as drone imaging). Our work attempts to use in-field temporally-dense images to monitor the number of canola flowers during the first days of the flowering season.

2.1 In-field vs. Controlled environment

Often, image-based phenotyping is performed under controlled environments, such as greenhouses or laboratories. This approach usually presents some advantages over performing in-field image-based phenotyping.

Image acquisition is typically performed under controlled conditions. In-field conditions can present high humidity levels, which can generate condensation on the lens of the camera or intense light reflections that prevent us from having a reliable image of a plant, thus interfering with the image processing procedures. For indoor imaging, these conditions can be controlled so that the images obtained are optimal.

Controlled environments also allow placing the plants in front of flat-coloured backgrounds to avoid the effects of undesired objects during the image processing processes. Additionally, plants can be placed at a certain distance from each other so that they do not overlap in the images. By doing this, individual plant analysis can be performed on each of them. In the fields it is common to find plants growing next to each other, making it very difficult to differentiate them when they grow. However, in a controlled environment, plants can be grown in individual pots, which can be separated when taking an image of them.

Ground truth acquisition is another advantage that controlled environments have over in-field phenotyping. For example, it is easier to manually count elements in plants (e.g., flowers or fruits) on plants that are growing in small pots than those growing in a large field growing along with each other.

2.1.1 Controlled environment

Controlled environments allow scientists to arrange the plants under study as desired (e.g., grids of fixed-sized cells, each containing a small pot) and take advantage of robotic arms facilitate the automatic acquisition of data and monitoring tasks, [25].

A different approach taken in controlled environments is to arrange the cameras and sensors in a specified position and fixed location, while the plants are moved with the help of a conveyor belt to go through these cameras and sensors, [27]. Illumination and image background in the pictures taken can be controlled to maximize the performance of the proposed methods, [10, 18].

Black or white backgrounds are often chosen to prevent external elements from interfering in the data analysis process. In other fields, such as microscopy, where the image processing methods are very similar to those used for plant phenotyping, count of elements also take advantage of having a black background, [17, 21].

Having plants separated from each other in pots allow for multi-view image capture, [26], which can be used for reconstruction of plants or movement tracking of leaves. However, performing these image acquisition techniques in the fields, where there is usually a higher number of plants growing close to each other, are often cumbersome due to a high overlap of plant leaves or flowers.

In some cases, plant phenotyping procedures need to take an image of other parts of the plants, such as roots, [29, 22], and requires to be performed in controlled environments where visual access to these elements is needed.

2.1.2 In-field

In-field work has the advantage of studying the evolution of plants under real environments, where weather, illumination and other factors are uncontrolled. Work using this approach is not as popular as indoor plant phenotyping. One of the main reasons is the challenge of overcoming natural effects, such as direct sunlight illumination, [36], which affect the detection of flowers or fruits directly, thus leading to a lower accuracy than experiments run in controlled environments.

The growth and arrangements of plants cannot be controlled either. It is common to find overlapping between plant elements (e.g., flowers or fruits), thus having some of them partially-hidden, which means that some of those items will go undetected. Specific approaches and algorithms are sometimes needed to overcome the overlapping problem, [47]).

Day and night times also affect the data acquisition process - images taken at night are often not usable, as the lighting is very low or completely null. Some methods rely on extracting pictures at certain times of the day, e.g., around noon, [6], when the sun is in an optimal position to minimize the effects of the direct sunlight illumination while maximizing the lighting of the scene.

2.2 High temporal vs. spatial resolution

Most popular in-field methods focus on having high spatial resolution data, which can be useful to cover large areas of a field or depict small features of a plant. However, there are some phenotypes with critical temporal characteristics, like the evolution of the number of flowers over time that spatially-dense data cannot depict.

2.2.1 High spatial resolution

The *high spatial resolution* term represents the acquisition of images that cover large areas while maintaining a resolution high enough to analyze small portions of the surface covered. Drone and satellite imaging is popular for in-field phenotyping where large fields have to be covered. Some of the pictures taken by current geostationary satellites are publicly available for researchers, [32]. Often, these approaches are combined with the use of multi-spectral cameras. Images acquired by these can provide additional information that regular RGB cameras cannot capture, as different plants have different spectral properties. Besides the regular RGB cameras, aerial imaging often uses infrared or multi-spectral cameras, [53]. Alongside with aerial imaging, ground-based platforms with multi-spectral cameras can be also used in the field, [30]. Near-infrared (NIR) cameras are often useful, [24], as healthy plants present a high reflectance in this spectral band. Often, some additional parameters are calculated from the intensity of various spectral bands. One of the most popular parameters is the Normalized Difference Vegetation Index (NDVI), [64], which provides information on whether there is live green vegetation in the picture under study.

2.2.2 High temporal resolution

A caption of temporal phenotypes is harder to perform in the field, as unsupervised continuous data acquisition is required, and the use of devices resilient to diverse weather conditions is crucial to avoid data loss. The study of temporal characteristics of plants is often performed under laboratories other controlled environments: Researchers can perform unsupervised data acquisition without the need to worry about external agents (e.g., cameras and sensors being exposed to rainfalls or strong winds). Sometimes, the methods implemented permit a low tolerance regarding errors in data acquisition. For example, the study of the growth of a plant leaf require to monitor closely the position of the leaves at any time, [11], or root plant evolution over time, [8]. Block matching algorithms can be used to keep track of the position of the leaves/root over time. The influence of wind on the plants could compromise the entire experiment. These methods can be also combined with multi-spectral cameras, [13].

Due to the similarity in the image processing methods or the time analysis of the data obtained, it is worth mentioning that time-lapse video and data analysis is popular in microscopy imaging. Position tracking, [16], and detection to predict the phenotype from the genotype, [43], are some examples of time-lapse analysis combined with image processing.

Researchers need to choose the rate of data acquisition carefully. High temporal resolution can translate into different data capturing rates depending on the experiment. For example, for an experiment that takes around one day, (e.g., some cellular-level processes) capturing an image every hour might not be enough, depending on what the researchers seek. However, for a process that takes one year, extracting an image every hour might seem excessive.

2.3 Image processing

One of the essential steps in our work is to detect canola flowers in images taken at the fields. This step infers the implementation of an image processing algorithm, which will do the vast majority of the work. This area has experienced an in-depth development up until today, providing a large number of methods and algorithms for different purposes. Much of the related work found that focuses on detecting elements from plants make use of custom-made image processing procedures. In the end, the image processing procedure that the researcher chooses depends mainly on the expected final result and the images on which said method would be applied. Images themselves depend on many other factors, such as the environment (e.g. outdoor vs. indoor location, daylight vs. night imaging) or the camera that is being used. However, it can be useful to review what previous researchers have implemented to detect elements (e.g., flowers, fruits) on various plant species. This section reviews some of the image processing techniques most commonly used by the literature found, which can serve as a basis to build up our image processing pipeline to detect canola flowers in the images we have extracted.

Regardless of the images with which we are working or the final desired outcome, all the image processing methods tend to follow the canonical image processing pipeline, whose steps can be described as:

1. Pre-processing: Consists of modifying the image so that the objects we want to detect are highlighted.

2. Segmentation: This step attempts to divide the image into areas of interest, and often, to get rid of those areas that are not important for our results.

3. Description: Extract numerical features that describe those areas in the image (e.g. feature extraction, or as it is in our case, detect the position of the flowers in the image).

4. Recognition: Assign labels or names to objects based on the output of the previous step (e.g. similar textures in an image tend to have similar features; therefore those can be assigned the same label).

2.3.1 Pre-processing

The pre-processing step comprises a wide range of operations, from intensity transformations such as thresholding or gamma transforms, to noise reduction. One of the steps that are considered as pre-processing is colour space shifting. Images, by default, are represented in RGB (Red, Green and Blue), which represents an image's intensity in the three primary light components. However, depending on the application, other colour spaces might be a better fit and thus improve the final result. Using RGB color space for image pre-processing can be used to depict elements whose colour reflects a high intensity of either red, blue or green, [65].

However, RGB also provides the image's intensity, which means that two images of the same object with different illumination will provide very different values in the RGB colour space. It is usual to find related work that uses a colour space where the luminance is separated from the colour description or *chrominance*. *HSV* (Hue, Saturation, Value) colour space codes the chrominance in its first two coordinates while the third is used for luminance: The Hue coordinate uses an angle from 0 to 360 degrees to represent which colour a pixel has, while the Saturation coordinate describes the dillution of the colour described by the Hue component. That is, maximum saturation means that the colour is pure (i.e., it is not dilluted), while minimum saturation means that there is no colour (i.e., the pixel will represent a scale of gray determined by the Value coordinate). Shifting from RGB to HSV can be very beneficial for images where the foreground has a colour that makes it very distinguishable from the background, [54, 57].

Another popular color space is *CIELab*. Similarly to HSV, one component represents the luminance of a pixel, while the other two are chrominance. In this case, the a component represents the green versus magenta intensity, while the b component shows yellow versus blue. The luminance channel can provide high intensities for white objects, [59]. A similar approach would be to use *YCrCb*, which provides red difference and blue difference in its chrominance components, [19].

As mentioned earlier, pre-processing can mean intensity transformations. One common operation is *thresholding*, which separates foreground from background given a threshold value based on each pixel's intensity. There are different approaches, such as global thresholding (i.e., the same thresholding value is applied to the entire image) or local adaptive thresholding (i.e., the threshold value varies across different regions of the image).

One popular thresholding algorithm is Otsu, [44]. It is designed to be applied to those images that produce a bi-modal histogram. This algorithm computes the optimal intensity that separates pixels belonging to each of the histogram's modes, [12].

After performing thresholding, it is usual to have noisy edges or small groups of pixels that can interfere in the future steps. Some methods utilize morphological operations such as erosion or dilation, [15], to smoothen sharp and noisy edges or to eliminate small clusters of foreground pixels that made it through the thresholding process but should be considered as background, [57, 12].

2.3.2 Segmentation

Some applications require dividing the image into two or more groups of pixels based on their properties. This approach is called *clustering* and attempts to assign each pixel to different groups based on a given feature (e.g., pixel's intensity) and a specified metric. The intended result is that pixels that present similar features should belong to the same group or cluster. One popular clustering algorithm is *K-means clustering*. It is an example of unsupervised clustering (i.e., does not need ground truth collection) where each element is assigned to a cluster based on its similarity with that cluster's centroid. The type of features used for elements in flower detection and counting are mostly colour, due to the contrast they present, with their background, [62, 59]. Two-dimensional clustering (i.e., using two features per element) can be used to determine the similarity between two pixels, using features such as colour and proximity to cluster the pixels, [5].

Besides presenting a distinct colour, often flowers have a certain shape, such as circular. Some methods were designed to identify certain shapes on a segmented image. Regarding flower detection, the most popular algorithm is the *Hough transform*, designed to detect elements of an *a priori* known shape. Even though the method was originally designed to detect straight lines, it is often used to detect circles or ellipses, [59, 38]. The *Hough transform* scans all the image for elements of a specified shape, transforming it into an N-dimensional space that will present notably high values in the positions where an object is detected. In the example of circle detection, there are usually three degrees of freedom (the radius of the circle and the X and Y coordinates of its centre); therefore it transforms the image into a 3-dimensional space where local maxima represent the centre of a possible circle.

Watershed algorithm is another segmentation algorithm that can be used to extract elements of interest from the background, not only for plant phenotyping [42], but also for other image processing applications such as microscopy imaging [16]. The idea of this algorithm is to interpret the gradient magnitude as a topographic surface, where high gradient areas (which correspond to local maxima) are region boundaries. If the gradient magnitude image is drawn in a 3D plot, the region boundaries correspond to dams, and the basins that the dams create are the segmented objects.

Another algorithm to separate an image in different regions is Region growing. This method starts from a seed point (or a group of seed points) that can be automatically selected, and grow a region around these seed points based on the similarity between the neighbouring regions and the seed points. This is, neighbouring regions that are similar to the seed points are added to the region. A similarity criterion needs to be specified in the method, which can be based on the colour or surface properties [45]. Sometimes, this method can be used after classification to correct regions that have been misclassified [46].

After performing some of the previous steps, there might be disjoint elements in the foreground that need to be identified and counted. Related work has showed that detecting connected components has been useful when the elements to be detected appear as disjoint groups of foreground pixels, [37, 34]. In some cases, flowers fulfill this description after performing thresholding, thus detecting and labeling them can be adequate methods to count them [38].

2.3.3 Description

Once segmentation has been performed, some applications require the elements of interest in the foreground to be assigned numerical values and descriptors so that they can be distinguished. Feature extraction algorithms, such as SURF, [9], or SIFT, [33], are very common. These methods can extract numerical descriptors to uniquely identify elements of an image (e.g., the cover of a book or the texture of a piece of cloth). In plant image processing, they can be used to describe various parts and use those descriptors to identify the plant. However, other features can be computed that describe colour, shape or area, that will be used for plant recognition, [50], [31].

2.3.4 Decision making

In combination with feature extraction algorithms, complex methods for classification are used. Artificial Neural Networks are found to be useful for feature classification. They are designed to predict an output given an input variable. Neural networks require a training or learning stage where input-output pairs are provided, and usually, a large dataset of training data is needed to obtain successful classification results. These methods are commonly used in plant identification, which can be based on its leaves' shape properties, [14, 31], or plant disease detection based on colour and texture features, [28]. Support Vector Machines (SVM) are also a classification method that identifies a feature point as part of one of two classes. They can be used to separate different parts of plants based on their surface or colour properties [46].

Deep learning

In the recent years, deep learning techniques have become more popular for identification and classification purposes, among others. Typically, deep learning techniques require large datasets to be trained. The latest advances in technology allow us to process larger amounts of information in an acceptable amount of time, which permits researchers to implement deep learning techniques and achieve with them high accuracy values in their methods. Convolutional Neural Networks (CNN) are widely used for identification of plants [52], plant elements such as roots or leaves [49], or plant disease recognition [58, 40]. Deep learning techniques can be used as well to detect fine details such as wheat spikes [48]. These techniques usually achieve high accuracies compared to other methods, are fully automated and often they require little image pre-processing. However, the dataset used for training is critical for them. Some techniques might have much worse results using pictures extracted under different conditions than that of the images used for training [40] Additionally, as mentioned earlier, some of them require large amounts of images to achieve these high accuracy values.

2.4 Summary

Previous work has shown different image processing and machine learning processes applied to the analysis, detection and counting of plant elements. Despite the different approaches, all the image processing designs tend to follow the canonical image processing pipeline, described at the beginning of Section 2.3. Additionally, many methods have to face an initial pre-processing to overcome adverse effects in the images that appeared during the data acquisition step. These kind of challenges are common on in-field image acquisition, and some of them will have to be tackled by our method.

Chapter 3

Data acquisition

Our imaging experiment was conducted in collaboration with Agriculture and Agri-Food Canada (AAFC) at a canola breeding trial located near Saskatoon, Saskatchewan (Canada). The trial consisted of 56 *b. Napus* cultivars grown in adjacent $12m^2$ (2m $\times$ 6m) plots that were replicated in 3 blocks (see Figure 3.1).

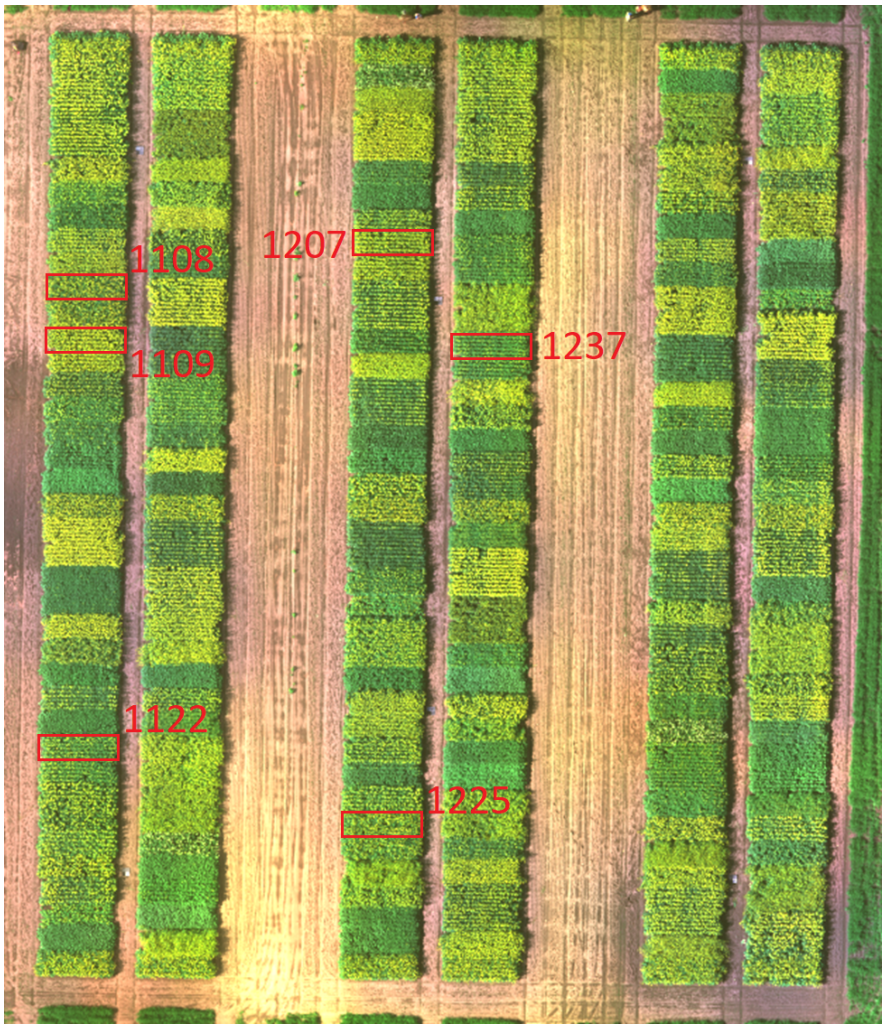

Figure 3.1: Aerial image of the entire canola crop taken during the flowering season showing the selected plots in red boxes (credit to Dr. Steve Shirtliffe).

We set up six Brinno TLC200Pro time-lapse cameras (Phase 3 Systems, Stuart, FL) to capture images of six different plots. The cameras were affixed to a round post 2 m high facing diagonally to the plot of interest. Each camera was situated one plot away from the one they were imaging. That way, given the field-of-view of the camera, the cameras could capture the entire boundary of the plot within its frame. A view of the camera set up can be seen in Figure 3.2, where the plots are delimited by orange and blue sticks.

The time-lapse cameras were initially configured to record non-stop before the first flowers bloomed. However, after a visual inspection, we found that roughly between 22:00 and 04:00, the images had little or no luminosity at all, thus not providing any additional information about the number of flowers. We also took into account the possibility that images during the flowering season at 04:00 or 22:00 could be a bit darker than those taken one month earlier at the same times, due to daylight time shortening. Therefore, we decided that for flower counting purposes, we would use only images taken between 05:00 and 21:00. Flower detection will be performed on all images taken within this time frame. Chapters 5 and 6 show how these images are used to determine the number of flowers on a given day.

We chose to image plots from 2 different accessions across the three replicated blocks with the intention of assessing repeatability. However, after collecting the data, we have been unable to assess said repeatability. The quality of the images was lower than we expected: some cameras produced blurry images or none at all for the early days of flowering; Section 3.1 discusses the different causes of these blurry or missing images, among other challenges. As for peak flowering days, there is high flower overlapping in pictures extracted from those days, and manually counting flowers in them can be very difficult; Section 5.3 describes how manual flower count was performed in the images and gives examples of early and peak flowering images.

Table 3.1 provides a short overview of the details of the configuration of the time-lapse cameras during the flowering season.

Table 3.1: Parameters of the cameras set-up

# of cameras	6
Image resolution	1280 x 720
Start/End recording time	05:00/21:00
Image interval	1 minute
Expected images per day per camera	960
Camera's height	2 m
Distance to plot	2 m

Figure 3.2: Image of one of the time-lapse cameras set up in the field.

The canola plants were seeded on May 27$^{\text{th}}$, 2016, and the cameras were installed in the field on June 21$^{\text{th}}$, 2016 (25 days after seeding). Cameras were removed on September 9$^{\text{th}}$, 2016 (105 days after seeding), after maturity and right before harvest. Through qualitative observation of the time-lapse videos, we found that flowers started to appear on day 38. The flowering stage of canola plants can last one month, [60], but full flowering can be reached within the first 10 days from flowering. Our method aims to detect flowers during the first days of growth and allow for an analysis of the flower density evolution over time. Thus, we decided to use and analyze images taken between day 38 and day 46 from seeding (referred to as day 0 and day 8 from flowering from now on). We will refer to this period as *early flower growth period*. Similarly, all dates mentioned from this point in the article must be interpreted as *days from flowering date* (unless specified).

During the early flower growth period, each camera was expected to capture 8640 images (960 images per day, which would be almost 52000 images in total).

3.1 Challenges

There were different challenges that we had to face during and after the data acquisition step. Most of these were not anticipated and made it hard to detect flowers in the time-lapse images accurately. In this section, we summarize said challenges and discuss approaches that can be taken to prevent them *a priori*, or overcome them once the pictures have already been extracted.

3.1.1 Direct sunlight on canola leaves

In some of the images, when the sky is clear, we can see that the sunlight hits directly the plants and the canola leaves are backlit. This causes that areas corresponding to canola leaves present a high yellow component. This can affect the entire flower detection process, as our method will rely on the characteristic yellow colour of the canola flowers to identify them. Figure 3.3 shows two different pictures taken on the same area at different times of the day. In each of the images, we applied a threshold on their CIELab's b component (which gives us yellow-vs-blue intensity) and painted in red every pixel that passed that threshold. Our intention with this is to show that the CIELab's b intensity of the canola leaves increases when direct sunlight hits them canola. However, when the sky is cloudy, we have diffuse lighting, as the sunlight is not hitting the field directly. This is the case of the left image in Figure 3.3, where the pixels that passed the threshold correspond to flower pixels. As we will see in further chapters, our method makes use of the CIELab's b component to detect flowers. This means that having direct lighting from the sun might interfere with the detection of yellow flowers, thus producing false positives (i.e., detecting flowers where there are none).

Unfortunately, there is not a proper way that this problem can be avoided during the data acquisition stage. Therefore an *a posteriori* solution has to be implemented.

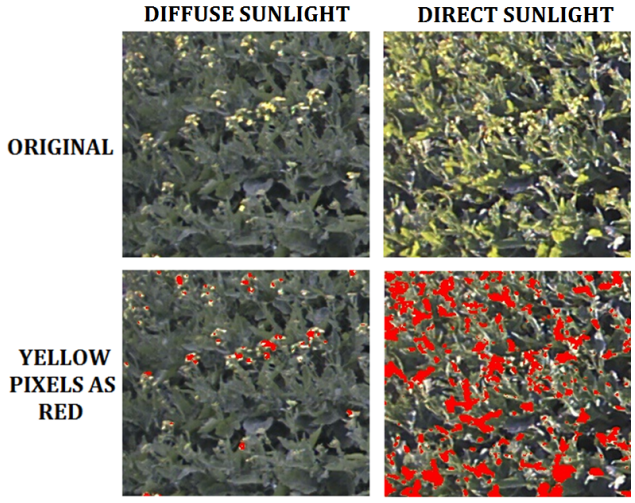

Figure 3.3: Example of effects of direct lighting over canola leaves (in red, the yellow pixels) on three different images of the same region taken on day 2.

3.1.2 Foggy images

We noticed the appearance of fog in images, especially at the beginning of some days. This problem affects our current pipeline in a way that different parameters are needed to detect flowers in foggy pictures than in the rest of the images. This means that our current implementation requires a way of classifying images based on the presence of fog and act accordingly based on such classification.

Figure 3.4: Comparison between an image with fog caused by condensation on the lens (left) and an image taken under normal conditions (right).

The appearance of such fog is due to the condensation that forms on the cameras' lens or the waterproof case. There is no easy way to avoid that, as we intend to avoid somebody physically going to the fields and check every camera one by one. This fog affects the image's b channel (from CIELab colour space) in a way that the flowers go undetected (i.e. the histograms are a bit compressed, therefore lowering the threshold wouldn't be enough) plus the images are blurry; thus any detection will be inaccurate.

Different lens protectors can keep the lens warm enough so that no condensation is formed. There exist anti-fog inserts that are placed inside the case, which absorb the moisture, preventing it from condensing on the case. A right combination of these proposed solutions (depending on the camera, some will apply or not) could be able to keep this effect from happening.

We were not able to predict this problem during the first season. Because images affected by fog require a different combination of parameters, we cannot run them through the pipeline along with pictures without fog in them. One possible solution is to tune the pipeline to be used with these images and run them separately from the rest. Related work shows how fog can be automatically detected in images, [7], and how it can be automatically removed, [23], [41]. However, having multiple images per day, and given that the condensation causing the fog in the images appears mainly at the early hours in the morning, an appropriate approach might be to remove them from the flower detection process.

3.1.3 Camera's angle and distance respect to the target plot

Due to the angle in which the cameras were set up, the size of the flowers within a single plot varies depending on their position. Nearby flowers look larger than those from plants situated further away from the cameras.

This effect is caused by the one-point perspective that the scene presents in the images. Further objects look smaller than those located nearby.

In the ideal case, the scene in the images would present an orthogonal perspective, which can be achieved by placing the time-lapse cameras over their plots of interest. This is the case of drone imaging. However, this set up can add other effects in the images, such as the shadow of the camera itself projected over the plants.

There exist projection transformation operations that can be applied to the images to convert a one-point perspective into an orthogonal perspective. However, this inserts distortion, as plants situated far from the camera will be stretched out. It is essential that the cameras be placed at an angle such that this perspective transformation inserts as little distortion as possible. For example, angles closer to 0 degrees respect to the camera's pole provide images with a perspective closer to orthogonal; therefore the projection transformation mentioned earlier would insert little distortion.

3.1.4 Missing data

During the data capturing process of the first season, we found that some of the cameras failed to record images for several days during the period of interest (i.e., early flower growth).

The Brinno TLC200Pro cameras were initially set up to record 24 hours a day, seven days a week (even though we only used images between 05:00 and 21:00 for flower detection). By design, they record time-lapse videos as an open file, and it is closed when the cameras are set to stop recording (either automatically or manually). However, in the case of full memory or dead batteries, the video file is never closed, and thus the video is lost.

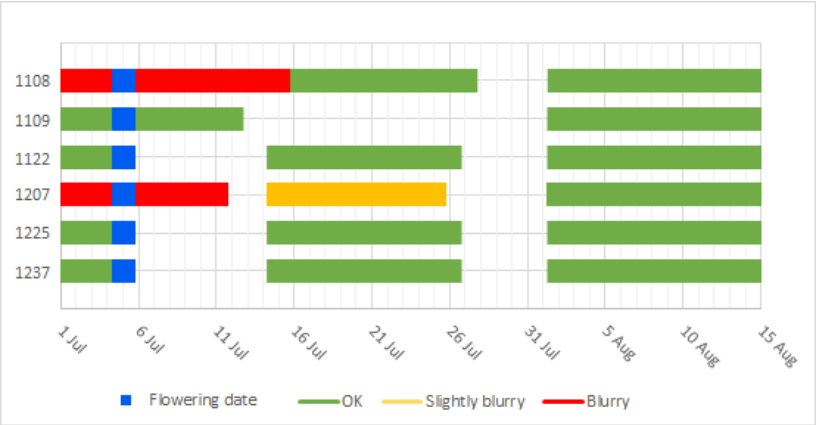

Figure 3.5: Data availability of each plot (Y-axis) across time based on the quality of the images.

This error was discovered later in the season, and the cameras were configured to start recording at 04:00 and stop at 22:00 every day. By doing this, the video files are always closed and complete by the end of each day. If the cameras run out of memory or battery, videos from previous days will remain stored in the memory card. However, it is highly recommended to estimate how long the cameras can record before the batteries die or the memory cards become full, to prevent data loss.

Knowing that the flower growth is a continuous function, it is feasible to estimate one day worth of data loss. Similarly, the derivative of the flower growth function is a continuous curve as well, which means that, if the number of flowers was increasing for the day N, it would likely be growing as well for day N+1. However, if the data is missing for several days, estimation becomes unreliable, as flower growth rate can quickly vary during early flowering days.

3.1.5 Blurry images

Some of the cameras captured a significant number of blurry images. Detecting flowers on them is difficult. Flowers can be hard to distinguish when presented in small clusters. Similarly, small isolated flowers could blend with the background, thus not being detected. It is usual to find a lower number of flowers detected in blurry images than in the sharp ones.

Figure 3.6: Comparison between an image taken under normal conditions (left) and a blurry image due to bad camera focus (right).

The Brinno TLC200Pro focus is manually adjusted on the lens with a small screw. A small screen in their back allows the user to see live what the camera is capturing. However, its reduced size makes it hard to determine whether the images are focused or not. Cameras must be correctly focused prior starting recording. While this can be a hard task for the Brinno TLC200Pro model, other cameras allow automatic focus, which would prevent any blurriness from appearing in the images.

It might be unfeasible to use image pre-processing to correct this problem when the pictures are too blurry. On the other hand, the number of flowers detected in blurry photos can be compared to that detected in sharp pictures. If it turns out that there exists a mapping between these two variables, a more accurate flower count can be estimated for blurry images.

3.2 Drone images

During 2016's season, a drone was flown every few days to capture aerial images of the canola plots. A Micasense RedEdge multi-spectral camera was attached to the drone to get the pictures of the crop. The camera generated each image's component as a separate image. Using a custom-made software, we aligned the multi-spectral components and corrected their intensities combine them in a colour image where we could detect canola flowers. Additionally, custom software was developed to stitch multiple images together to have a single view of the entire crop. Figure 3.1 is an example of the result of multiple drone images whose red, green and blue components were stitched,

After a drone flight, several images were extracted. These images had to advantages over time-lapse images:

- Orthogonal view of the plot. As we mentioned in Section 3.1, having a one-point perspective of the plot might cause having a different number of flowers detected at different points of the same plot. Drone imaging addresses this problem by providing an orthogonal view of the plots, where the distance between two flowers in the plot and the image is independent of the position of those flowers within the plot.

- Multi-spectral image. The Micasense RedEdge provides a near-infrared and red edge components that might be helpful for different image processing purposes. However, as our main focus was to work with time-lapse images, we have developed our method to work with RGB images and their representation in other colour spaces.

However, the main disadvantage of the drone images is that it could not constantly be flown to take many images on the same day. During 2016's season, the drone was flown only once every two weeks roughly. That means that we could not use these images to monitor the growth of the canola flowers, as canola plants can go from zero to peak flowering within two weeks.

3.3 Summary

The numerous challenges that we encountered during the data acquisition process have impacted our pipeline directly. Additional steps and measures had to be taken to be able to extract reliable data from the images. A recommended approach to solving some of the main problems that we faced is to use a more proper camera model: Blurry photos could be avoided by having a camera with automatic focus. Additionally, by having a higher image resolution, we might be able to address flower overlapping, and thus use images from peak flowering.

Due to data availability and quality, comparison of canola growth across plots became unfeasible. Additionally, looking at Figure 3.5, we can observe that after the flowering date, only three of the cameras produced images. However, the pictures taken by two of them were too blurry to perform an accurate analysis. As discussed in Section 3.1.5, flower detection might not be reliable on these images, and manual flower counting can be hard for the raters to generate reliable ground truth. Only one of the cameras (1109) produced sharp images during the first days of flowering. Based on that, we decided to use the images from that camera to manually and automatically count canola flowers.

Chapter 4

Pre-processing

There is information that can be extracted from the time-lapse images that will be useful for the future steps. Our approach uses the timestamp of each image to separate the images by day of extraction. Each image not only captures the camera's plot of interest but also parts of the crop that are not relevant for further image processing steps. Additionally, the histograms of the images can tell us important information about them such as how many yellow areas it contains. In this chapter we described procedures that attempt to address these issues, which are:

- Extraction of each image's timestamp.
- Definition of the boundaries of the plot of interest.
- Calculation of misalignment of the histogram of the CIELab's b channel from each image respect to others taken the same day.

4.1 Image and timestamp extraction

Brinno cameras produced a time-lapse video instead of image files. FFmpeg was used to separate the time-lapse videos into frames. Most cameras write the image's timestamp on the file's metadata, which is easier to extract. However, the Brinno TLC200Pro printed the timestamp on a narrow black band at the bottom of the image itself, which presented the following characteristics:

- The timestamp's format was **TLC200PRO yyyy/MM/dd hh:mm:ss**.
- This black band printed at the bottom was always 16 pixels high going from left to right of the image.
- All characters were white, and their shape was invariant.
- None of the characters were overlapping (i.e., there was always a vertical black line at least 1 pixel wide that separated each of them).

Based on these, we stored a copy of each possible character as an image. For a new image, the timestamp extraction process was as follows:

1. Keep the 16 bottom lines of pixels of the image (which would correspond to the timestamp).
2. Split them into groups of vertical lines, using as delimiter a vertical black line, which will provide each character as a single group of pixels.
3. For each group of pixels, calculate the Mean-Squared Error (MSE) with the images of every possible character we initially stored and assign this group the character that gives the lowest MSE.

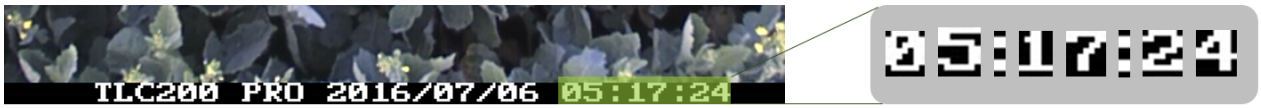

Figure 4.1: Illustration of an example of a timestamp and how characters are divided.

4.2 Plot region definition

We manually defined plot boundaries in the time-lapse images for each of the plots under observation once for the entire image segment. A simple graphical user interface (GUI) was developed that allows the user to specify the plot's boundaries. Figure 4.2 shows what this GUI looks like: A coloured trapezoid is displayed whose corners have to be moved using the mouse to their correct location. This GUI presents the user with an image of the plot of choice and has to move the corners of a trapezoid overlaid on the picture to the plot edges. For future steps, pixels outside the plot boundaries are ignored, as they do not contain any relevant information.

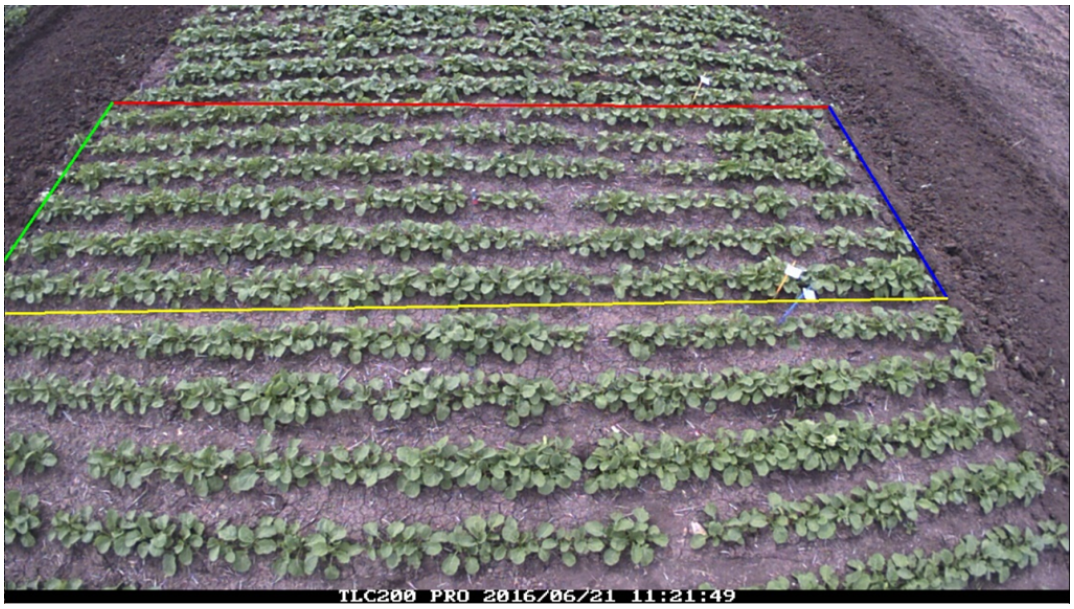

Figure 4.2: Illustration of the plot region definition GUI where the trapezoid is enclosing a single plot.

4.3 Histogram alignment

Previous work shows that working in a color space other than RGB can be beneficial for colour-based segmentation of objects such as flowers, [54], or fruits, [62]. Canola flowers are characteristically yellow, therefore to distinguish yellow pixels from the rest, we used CIELab color space, which provides a *yellow-vs-blue* channel that presents high values on yellow pixels and lower values in the remainder of the image.

Images taken by the same camera during the same day are expected to give similar histograms of this CIELab's yellow channel. However, we observed that these histograms have appeared shifted horizontally across images taken on the same day. Figure 4.3 shows at the bottom left corner the histogram of two pictures taken on the same day and plot prior histogram pre-processing. Their shape is similar, but one is shifted across the X-axis from the other.

Some processes, such as thresholding, require a pixel intensity value. However, due to these histogram shifts, each image requires a different intensity value, which would be calculated as a function of the shift amount. We calculated how much each of the CIELab's b channel's histogram is shifted (referred to as *histogram shift* from now on). The histogram shift is calculated as follows:

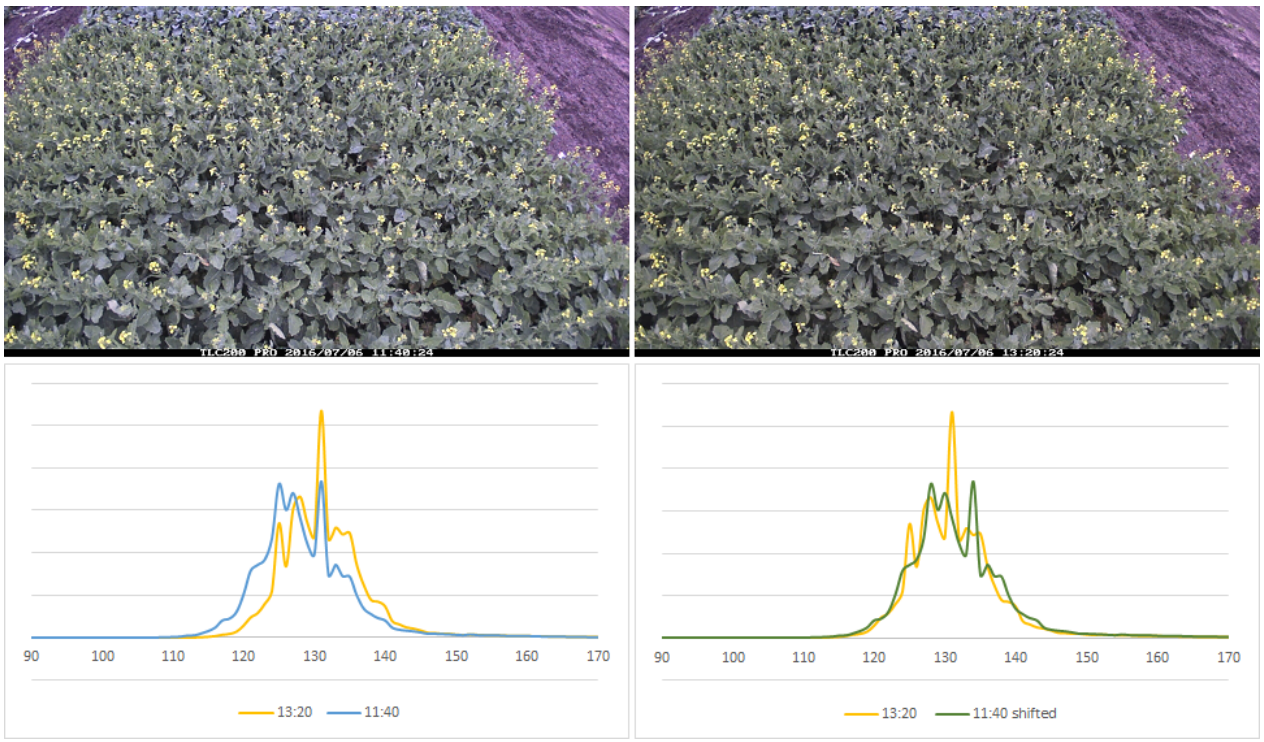

Figure 4.3: Example of the histograms of the CIELab's b channel of two images from plot 1109 taken on day 2 from flowering.

1. We group the images by day and plot. This means that the histogram shift of an image will be computed using only histograms from images taken on the same day and plot.

2. Calculate the histogram of the CIELab's b channel, $h_i(n)$, for every image of a plot in a single day, where n is an integer that can take the value of any of the possible pixel's intensities.

3. To be able to calculate the shift of the histograms of all images, we need to have a reference histogram, so that we can calculate by how much each histogram is horizontally shifted respect to the reference histogram. However, at this point, we do not have any information as to which histogram should be the reference (i.e., which histogram should have a shift of 0). So, for now, we randomly choose one histogram to serve as the reference, $I(n)$. It is possible that the randomly-chosen reference histogram might not be such a good reference (e.g., if $I(n)$ is significantly shifted from the rest of the histograms). However, this potential problem is addressed in further steps within this procedure.

4. Calculate the histogram shift, S_i, of each individual histogram, $h_i(n)$, respect to $I(n)$, by calculating the argument that maximizes the cross-correlation between $h_i(n)$ and $I(n)$:

$$S_i = \arg\max_x \sum_{\forall n} I(n) \cdot h_i(n - x) \tag{4.1}$$

5. We just calculated how much each histogram is shifted towards a reference histogram that we chose randomly. However, we there is a possibility that our reference histogram, $I(n)$ is heavily shifted from the rest. For example, let's say that all $10 \leq S_i \leq 12$ for every histogram. We can say that, if $S_i = K$ and $S_j = K + 1$, then the displacement between the i^{th} histogram and the j^{th} histogram is 1. In this situation, we assume that $I(n)$ is a bad reference histogram. We believed that these histogram shifts were occurred due to changing lighting conditions throughout the day. We assumed that the majority of the time, the lighting conditions were similar and thus we should consider those lighting conditions as the reference lighting conditions. That would mean that the majority of the images were taken under those so-called reference lighting conditions, which would mean that those images should be considered as a reference when calculating this histogram shift. This means that the majority of the images should have a histogram shift of 0, as a well-chosen reference histogram should have the same shift as the majority of the images (i.e., the mode of all the calculated histogram shifts should be 0). If the majority of the images had a histogram shift other than zero, we could think that our reference histogram was incorrectly chosen. Otherwise, it would be very coincidental that a large group of images were taken under different lighting conditions than the image of the reference histogram (thus having a histogram shift different from 0) and still had the same histogram shift among them. As we believe that the former situation is more likely than the latter, we decided to address the problem of having chosen an incorrect reference histogram. Given these assumptions, we calculate the final histogram shift of the i^{th} image as:

$$S'_i = S_i - M \tag{4.2}$$

$$M = mode(S_i), \forall i \tag{4.3}$$

By doing this, we make the majority of the images to have a histogram shift of 0.

Figure 4.3 shows an example of two images that were taken on the same day and plot (upper half). These images have histograms that are slightly horizontally shifted (lower left image). By using the procedure described above, we calculated that the displacement between them is 4 units. While the bottom-left side of the image shows the original histograms before applying any histogram shifts, while the bottom-right side of the figure shows the histograms after applying to one of them the histogram shift calculated respect to the other.

4.4 Results

4.4.1 Timestamp extraction

From the entire data set, we decided to use 30 time-lapse images where the timestamp band was perfectly legible. The ground truth extraction was done by manually creating a list with the name of each file and the timestamp that they had in the expected format. The code designed for this was written in Python, taking an image array at the input, and returning a **datetime** instance with time zone as "America/Regina."

For non-corrupted images, the success rate was 100%. However, a group of images were corrupted in a way that either the timestamp was coloured, or completely disappeared (see Figure 4.4). The success rate for these was 0%. However, as we mentioned, the period between images was always 60 seconds, which means that the timestamp of a corrupted image could be estimated using either the previous or the next picture.

Figure 4.4: Example of corrupted images with a timestamp with different colour (left) or completely erased (right).

Percentage of total success/failure

The total dataset contained 667618 images. Our code was able to parse the timestamp of 87.21% of these. The timestamp of the rest of the pictures could be estimated from those taken earlier or later, thus being able to extract the timestamp of the 100% of the images used for this experiment.

4.4.2 Histogram alignment

Histogram shifts are calculated per day and plot. This means that the histograms are aligned with those from images taken on the same day and by the same camera. To show the results of our implementation, we have performed histogram alignment on the images taken on plot 1109 on day 1 from flowering. From this subset of images, we have chosen 4 of them and plotted their CIELab's b histograms in Figure 4.5. The left side of the figure shows these histograms before applying any histogram shift where, for example, we can see that the green histogram is heavily displaced from the rest. On the right side of the image, we show the same histograms after being shifted by their calculated histogram shift. In this case, we can see that the green histogram aligns better with the rest of the histograms, as we expected. In Figure 4.6 we plotted the histogram of all the shifts of the images taken on plot 1109 on day 1. In it, we can see that the majority of the images have a histogram shift of 0, as our method forces in the last step. In this example, it would be very unusual that the majority of the histograms had a shift different than zero. As we explained earlier in this chapter, it would be very coincidental that the majority of the images (in this case, the 86% of all the images) were taken under lighting conditions different from the so-called reference lighting condition that caused them to have the same histogram shift with a value other than 0. In such case, it would make more sense to think that the choice of the reference histogram, $I(n)$, was not correct and that problem should be addressed (as the step 5 in the histogram shift calculation method does).

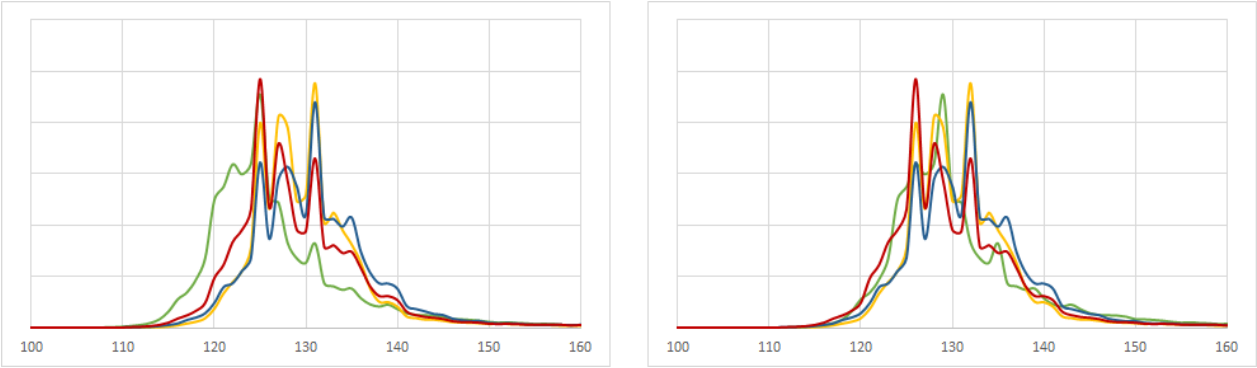

Figure 4.5: Histogram of CIELab's b channel of 4 random images from day 1 from flowering, plot 1109. Left: Original histograms. Right: Histograms after applying shift

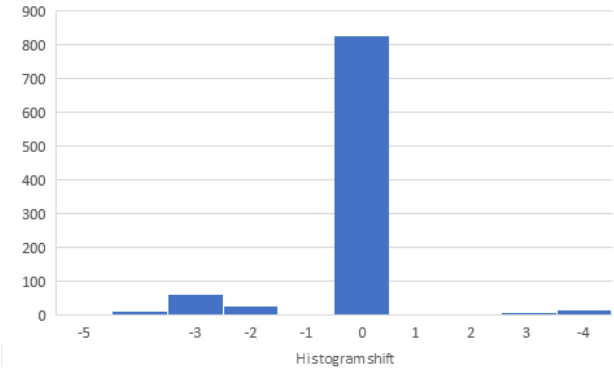

Figure 4.6: Histogram of the calculated shifts, S_i for the images taken in day 1 from flowering on plot 1109 before applying.

Chapter 5

Flower detection pipeline

In this chapter, we show the process that we have developed to detect canola flowers in time-lapse images. First of all, we need to operationalize a canola flower. This is how we identify a canola flower in our images, what unique characteristics it has that can be used to identify them, etc. Once we have defined the concept of a canola flower, we can start looking into the different possibilities we have to detect and count them in the time-lapse images automatically. There are various image processing approaches that we discussed in Chapter 2 whose strengths rely on the nature of the objects we want to detect.

5.1 Operationalizing a canola flower

To define the concept of a canola flower in a time-lapse image, we started by taking a look at the images from the early flowering season. Figure 5.1 is an example of an early flowering season time-lapse image. In this picture, we can observe the flowers separated from each other, which can help us with the flower operationalization process.

We could say that the main visual characteristic of canola flowers is their colour: There is usually a notable contrast between the yellow colour of the flowers versus the green chrominance of the canola leaves. Therefore we could start the definition of canola flowers in time-lapse images as yellow areas over green background.

The second characteristic that we can observe in Figure 5.1 is that the flowers usually appear as small roughly-round yellow shapes over green background. While the colour makes them distinguishable, it does not seem possible to detect unique shape features on the flowers. Additionally, the size of these flowers does not look very large: by observing it directly, we could say that the size of a flower could range from a couple of pixels (5 pixels) to roughly 20 pixels from side to side.

Given these observations, we can determine that, to detect yellow flowers in our time-lapse images, we shall look for round-like yellow shapes whose size from side to side ranges from 5 to 20 pixels roughly. However, using this concept of canola flower in time-lapse images can have some disadvantages:

Figure 5.1: Example of a time-lapse image of a canola plot taken during the early flowering season.

- We have discussed earlier in Chapters 3 and 4 the effect that lighting differences and direct sunlight can have on the images. As we already discussed in said chapters, these phenomena alter the colour of the image and could difficult the detection of yellow areas.

- We have a similar situation with foggy images, also discussed in Chapter 3: The colour of the image is altered, forcing us to address these images separately.

- When some flowers grow close to each other, they can overlap in the images. This could mean that these overlapping flowers would present as large yellow areas, and depending on how our method is implemented, they could be interpreted as one large single canola flower or many canola flowers.

5.2 Method

There are multiple popular approaches for detecting elements in images after pre-processing is done. For example, feature extraction has proved to be very useful when detecting unique characteristics of the objects of interest, such as shape or colour, [50], detecting circle-alike shapes using Hough transform, [59], or applying feature extraction methods such as SURF, [9], applied to chrominance channels, [35]. In our case, because we have defined canola flowers as yellow areas, we will attempt to detect flowers by using their colour (yellow) and shape (round-like) features. The size of the flowers in the images is a limiting factor: we believed that methods such as the previously mentioned Hough transform would not perform well, as the shape of the flowers is not precisely circle, but it can present as a circle, ellipse, etc.

The objective of our method is to detect small yellow areas. The CIELab colour space provides in its b coordinate the yellow-vs-blue intensity of each pixel. This means that, in this channel, yellow areas will have a higher intensity than non-yellow regions. We have defined the shape of the flowers as round-like. However, another term that, in our opinion, could apply to the canola flowers is *blob*. We could use a blob detector on the CIELab's b channel to detect yellow blobs in the image. However, during the development of the method, we felt the need to add extra steps to reduce the appearance of false positives/negatives.

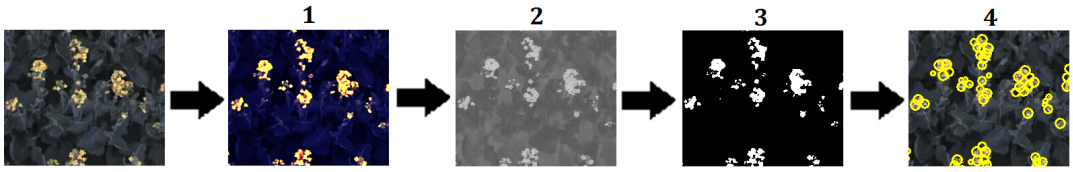

Figure 5.2: Flower detection example over a fragment of a time-lapse image. Steps: 1. RGB Gamma Transform, 2. Color space shift, 3. CIELab Intensity mapping, 4. Blob detector.

The steps of our image processing pipeline and their details are described below:

1. **RGB Gamma Transformation:** The first step applies a gamma transform to the Red and Green channels from the RGB colour space. The original objective of this stage was to reduce the effect of the direct sunlight hitting the canola plants that we mentioned in Section 3.1.1. Using image processing software (e.g., Adobe Photoshop) we tried to manipulate the Red, Green and Blue channels attempting to find a transformation that would reduce said reflections. We observed in the CIELab's b channel that areas directly illuminated by sunlight and flowers had similar intensities. That is why we believed that we could reduce the reflections intensity by using a different colour space prior manipulating the CIELab's b channel. We found that the Gamma Transform in the R and G channels reduced the intensity of some reflections. The Results section of this chapter shows that the accuracy is higher with this RGB Gamma Transform (see Section 5.4.2.2). We attempted to find the transformation in the CIELab that produced the same effect. To do so, we obtained the CIELab's b channel of an image before and after applying the RGB Gamma Transform. Figure 5.3 shows the intensity relationship between these two images in the Red channel (left) and in the CIELab's b channel (right). The transformation between the two images in the CIELab's b channel does not correspond to a known curve; thus we decided to apply the Gamma Transform using the RGB colour space instead of using the CIELab colour space.

   We observed that a Gamma Transform in the Red and Green channels helped reduce the effect of direct sunlight in a handful of images taken during sunny periods. We decided to incorporate it and, as it is shown in the Results in Section 5.4, the accuracy increases having this step in the pipeline.

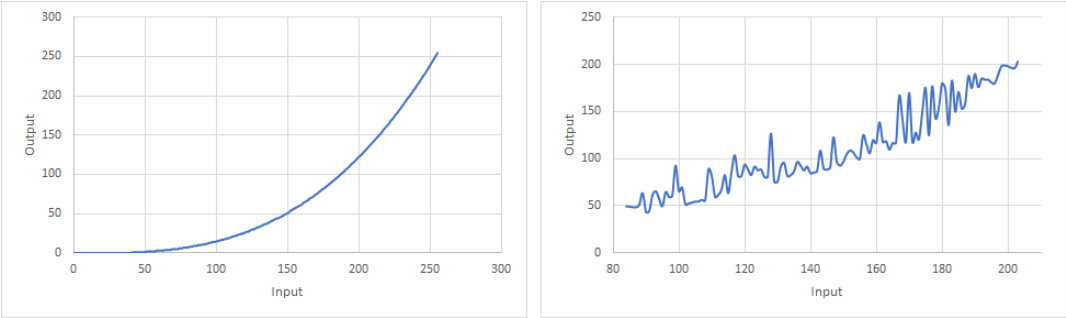

Figure 5.3: Pixel-wise intensity relationship in the RGB's Red or Green channels (left) and in the CIELab's b channel (right) before and after applying RGB Gamma Transform.

The Gamma Transformation is defined by the formula below:

$$y = 255^{1-\gamma} \cdot x^{\gamma} \tag{5.1}$$

where x and y are the input and output intensities respectively. A scaling factor of $255^{\gamma-1}$ ensures that $0 \leq y \leq 255$. Figure 5.5 shows the input/output relationship in this step, and the visual result of this step shows the flowers highlighted in yellow while leaving the background as blue (see step 2 of Figure 5.2).

2. **Color space change:** As we mentioned before, yellow pixels are well distinguished in CIELab's b channel. In addition to this, the previous step transformed all the background pixels to blue, leaving the flower pixels yellow, which means that this flower-vs-background distinction should be clearer in said colour space. This step converts the image from RGB to CIELab colour space representation and extracts the b channel. As mentioned earlier, the **b** coordinate can reach negative values. However, OpenCV transforms these intensities so that the output values will fall within the range $[0, 255]$. Figure 5.2) shows the result of this operation.

3. **CIELab Intensity Mapping:** We perform an intensity mapping in the CIELab's b channel, with which we attempt to increase the intensity of the yellow pixels, while darkening the rest of the colours. A sigmoid curve (Figure 5.4) applied to the CIELab's b channel could have this effect. The formula of a sigmoid curve is:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{5.2}$$

However, we need to be able to move this curve in the X axis and take into account the histogram shift calculated in Section 4.3. Additionally, we need to apply a scaling factor so that $S(0) = 0$ and $S(I_{max}) = I_{max}$, where I_{max} is the maximum possible pixel intensity. After adding these adjustments to the sigmoid formula, the resulting equation is:

$$f(x) = \frac{255}{1 + e^{k \cdot (t - S_i - x)}} \tag{5.3}$$

where x is the input intensity of a pixel, S_i is the histogram shift of the ith image (calculated in 4.3). In the formula above, the parameter t acts as a threshold, determining which pixels will be considered yellow and which will not, while k defines how aggressive the mapping will be. A scaling factor is applied to ensure that $0 \leq y \leq 255$. A sample of what the image looks like at the output of this operation is shown in step 4 of Figure 5.2. In Section 5.4 we compare the performance of the Sigmoid mapping step with other intensity mapping approaches to validate our choice.

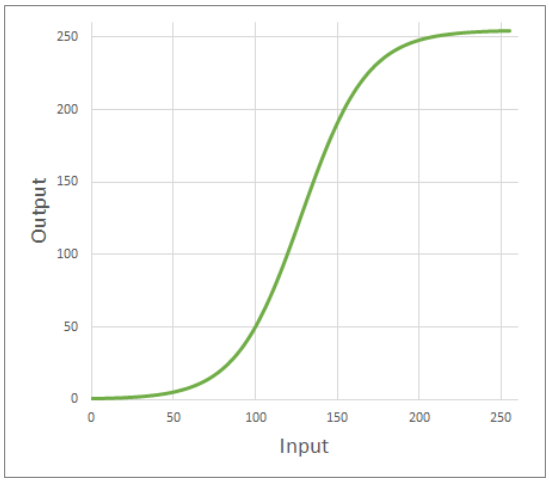

Figure 5.4: Illustration of a generic sigmoid curve shifted horizontally to fit within the $[0, 255]$ range.

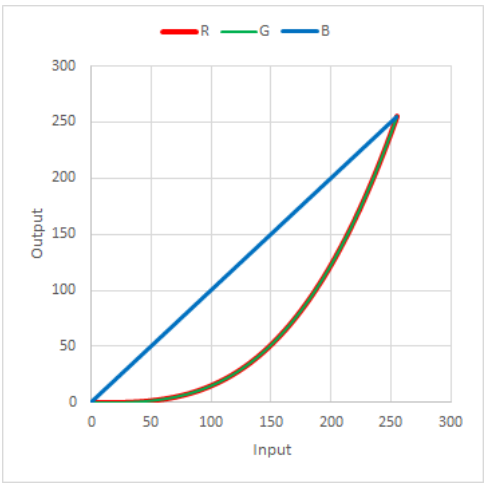

Figure 5.5: Intensity transformation performed on the RGB color space on the Gamma Transform step.

4. **Blob detection:** At the input of this step, the flowers appeared as white blobs over a black background. We ran SciKit's blob detector implementation, [61], [55]. This approach offers three different variants: Laplacian of Gaussian (LoG), Difference of Gaussian (DoG) and Determinant of Hessian (DoH). SciKit's implementation provides not only the position of the blob but also the radius of each of the blobs, which can be used to calculate its area (i.e., an approximation of the area of the canola flower represented by that blob). In Section 5.4 we present an analysis of the performance of each of the three approaches with different parameter combinations. They show that the DoH has produced better results; therefore we decided to use that approach in our pipeline. An example of the flowers detected by the blob detector is shown in step 5 of Figure 5.2.

To detect multiple flowers overlapping, we could look for a relationship between the blob size and the actual number of flowers in the detected blob. If our pipeline is not able to distinguish many flowers within one single blob, we would expect that larger blobs correspond to clusters of flowers, rather than one big flower. However, as difficult as it is for our pipeline to detect multiple flowers in the same cluster, so it is for a rater to manually count many flowers that are overlapping in the image, therefore it is hard to have reliable ground truth to test this process of separating overlapping flowers. Also, having temporally dense data does

not help us in this situation, as we cannot track the position of the flowers over time. From image to image, the location of a flower in the image can change drastically due to wind, and when the number of flowers is too high, it is very difficult to keep track of a single flower over time.

5.2.1 Blob detection approaches

As we mentioned earlier, there are three different approaches for blob detection that we can use: The Laplacian of Gaussian (LoG), Difference of Gaussians (DoG) and Determinant of Hessian (DoH). They all process a two-dimensional image to look for blobs described as local maxima in intensity. The DoG and LoG approaches follow a similar process:

1. Blur the image using a Gaussian mask.

2. Repeat the first step using different σ values, which represent the variance of the Gaussian mask.

3. Stack the resulting 2D images to form a three-dimensional image.

4. Find local maxima points in the resulting 3D image, which will correspond to blobs.

The difference between DoG and LoG resides in the mask they use. While the LoG uses a 2D Gaussian function as a mask, DoG uses the difference of two Gaussian functions to blur the images. In Figure 5.6 we can see a graphical representation of each of the masks, in which the left image corresponds to the LoG mask, and the right side of the image shows the DoG mask.

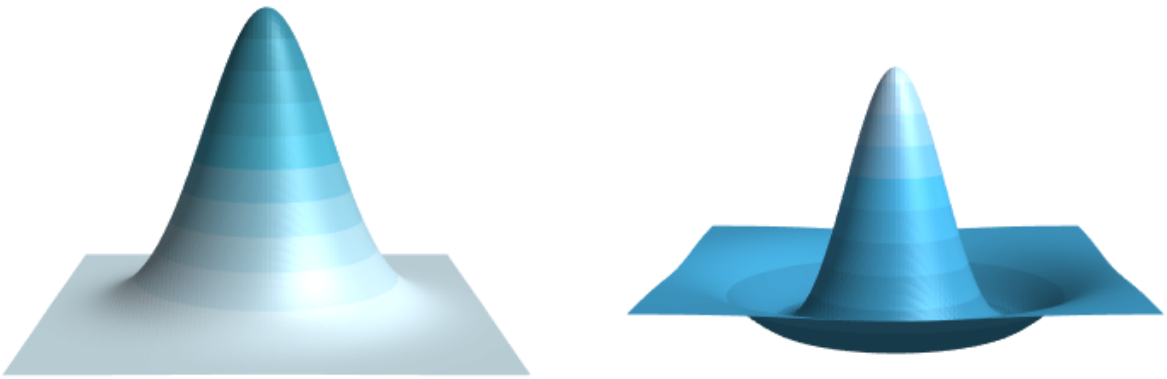

Figure 5.6: Graphic illustration of the masks used by the Laplacian of Gaussian (left) and the Difference of Gaussians (right).

The DoH approach follows a different procedure:

1. For each pixel, we calculate the Hessian matrix at that pixel's position.
2. For each Hessian matrix, calculate its eigenvectors and eigenvalues.
3. In a given location the eigenvector at that position with the highest eigenvalue determines the direction of the greatest difference in intensity, while the eigenvector with the lowest eigenvalue points in the direction of the lowest difference in intensity. If we represent the intensity of a pixel at the position (X, Y) in the third dimension (i.e., the height of the image at (X, Y)), one of the eigenvectors points to the direction with the highest curvature and the other points in the direction of the lowest curvature.

Once we have done this, we can determine if there is a blob at a specific location by looking at the eigenvalues. If the absolute value of both eigenvalues is higher than zero, it means that the curvature is high in the direction of both eigenvectors, which indicates the presence of a blob. On the contrary, if the eigenvalues are close to zero, means that the image is almost flat at that point, thus indicating that there is no blob. Additional parameters, such as a threshold applied to the eigenvalues' magnitudes, can be used to determine whether a low-intensity blob can be considered a blob or not.

Figure 5.7 provides a graphical representation of the intensity of a 2D image, where the third dimension represents the intensity of the image at a given location (i.e., high locations correspond to high pixel intensities, while low locations represent low intensities). In that case, we can see a blob in the position (-10, -10), where both eigenvalues have a similar magnitude, indicating that the curvature is the same in both directions. On the flat surface, both eigenvalues will have a value of zero, indicating that the curvature is flat and equal in both directions.

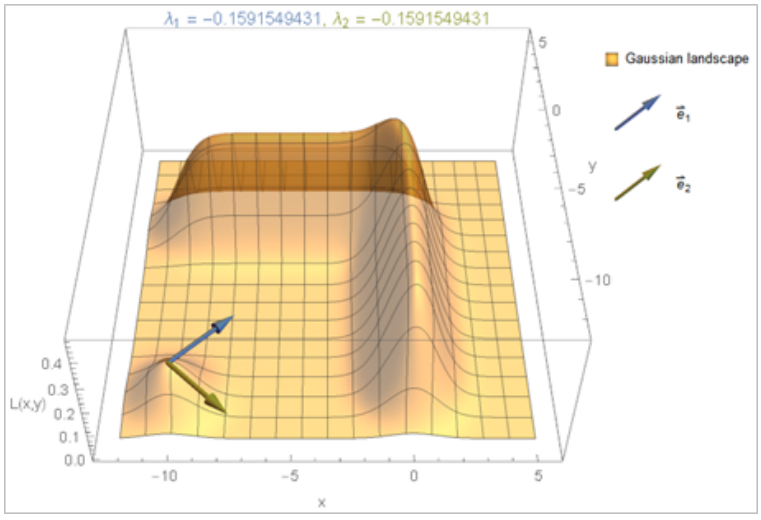

Figure 5.7: Graphic illustration of the intensity of a 2D image and the Hessian matrix's eigenvectors calculated at a given point in the image. (Credit: `https://milania.de/blog/Introduction_to_the_Hessian_feature_detector_for_finding_blobs_in_an_image`)

5.3 Ground truth acquisition

To verify the flower count values extracted by our pipeline, we have performed a manual count on a small subset of images taken this day. Five people were asked to perform once this manual count. They were asked to count the flowers in these images. The instructions the users received were the following:

- The user was required to count all the flowers they were able to see within a region that was marked on the image.
- The user was given an example of what was considered a flower (Figure 5.8 shows an example of this).
- In case of ambiguity (e.g., when flowers grew too close to each other) they were instructed to use their judgment to decide the number of flowers they were seeing. Figure 5.9 shows an example of this situation and the result of two different users.

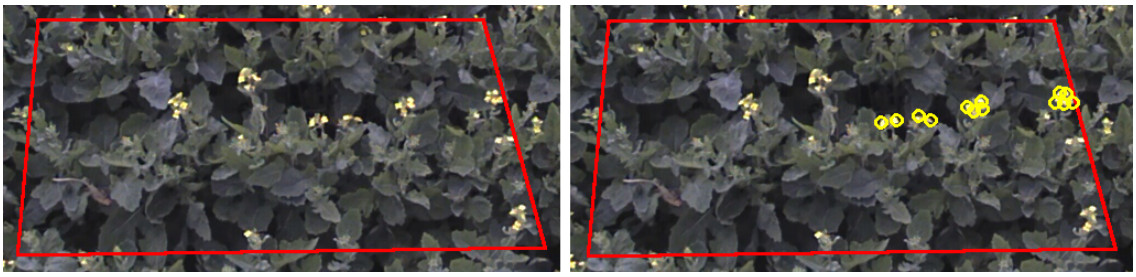

Figure 5.8: Illustration of what the user is presented (left) and some flowers rounded to serve as example (right).

Figure 5.9: Example of compact clusters of flowers (left) and how two different users have detected them (middle & right).

The users were provided with a simple Graphical User Interface (GUI) where an image is presented with a region marked with red boundaries. The user is asked to click on every flower they can distinguish within that region and press any key when they are finished. The images were presented in random order to avoid bias from previous counts.

Manual flower counting can be tiresome for the users if the number of images or the flowers is large, which can cause the user to provide an unreliable manual count. To avoid this situation, the region in which the user was asked to count flowers consisted of a small portion of the entire plot, where the number of flowers could range from 0 to 80 (depending on the image's day of capture).

We used 20 images per day from day 0 to 5 from flowering taken on plot 1109 (i.e., a total of 120 images) to extract the ground truth. The images were hand-picked so that the scene had a diffuse illumination (i.e., there was no direct sunlight over the canola plants) to make it easier for the rater to manually count flowers. From the five users that participated, three of them were Computer Science students, while the other two were Biology students. To calculate the inter-rater reliability, we decided to calculate the *intra-class correlation* (ICC) which can be used to estimate inter-rater reliability when the data is continuous (i.e., quantitative) rather than classified (i.e., qualitative) data [3]. All our raters have counted flowers in the same images, and we can consider that they represent a sample of the population of raters, as many other people could have counted flowers in these images. Based on this, we decided to use class 2 of ICC (also known as ICC(2)). The intra-class correlation calculated was 93%, with lower and upper bounds of 87% and 95% respectively, using a confidence level of 95%.

5.4 Results

5.4.1 Error measurement

To validate our results, we will compare the flower count at the output of our pipeline in a group of images against the manually annotated number of flowers. To provide a numerical, we calculated the average relative error and correlation between the manual and automatic flower count.

The relative error between automatic and manual flower count can be calculated as follows:

$$error = \frac{|x - y|}{x} \tag{5.4}$$

being x and y the manual and automatic count respectively. However, to solve the zero-denominator problem, we use the following formula:

$$error = \begin{cases} 0 & \text{if } x = y = 0 \\ \frac{|x-y|}{max(x,y)} & \text{otherwise} \end{cases} \tag{5.5}$$

Finally, the average relative error over all the images tested is:

$$Avg.Error = \sum_{i=0}^{N} \frac{error_i}{N} \tag{5.6}$$

where $error_i$ is the relative error calculated for the i^{th} image, and N is the total number of images used.

To calculate the correlation between two sets of values X and Y we used Pearson's correlation coefficient, for which the Python library *NumPy* provides functions to calculate.

5.4.2 Parameter tuning

Almost every step of the pipeline allows parameter tuning. The entire configuration contains a total of 8 parameters, distributed as follows:

- **Gamma Transform:** 1 parameter.
- **CIELab Intensity mapping:** 2 parameters.
- **Blob detection:** 5 parameters.

The pipeline described in this chapter can analyze each image independently from the others. Therefore, we designed our code to distribute the flower detection processing of each image over 40 cores. We found that our design needed around 20 seconds (at the best case) to process 120 images (i.e., the ones from which ground truth was extracted) given a single combination of parameters. To test 5 different values for each parameter, we need to try over 390000 combinations. The blob detection step allows three different approaches (DoH, DoG and LoG), which means that we will have to test these combinations for each of the three methods. This leaves us with over 1 million combinations, which will take 270 days to process in the best case.

Tuning strategy

We decided to do a 3-step parameter tuning described as follows:

1. Tune the Gamma Transform and the Intensity Mapping steps together as one (3 parameters in total). For this step, we used the DoH blob detection approach as it has been reported to be the fastest. The parameters used for the DoH blob detector were their default except for $Max.Sigma = 7$, $Min.Sigma = 3$. In future steps, we will use the three different blob detection approaches and compare their performance.

2. Tune the Blob Detection step. Here we will use the three approaches (DoG, DoH and LoG) and tune them separately. For this steps, we have used the Gamma Transform and Intensity Mapping parameters that produced the lowest error in the previous step.

3. For each of the three Blob Detection approaches, we tune the Gamma Transform and the Intensity Mapping steps again. In this case, we use the Blob Detection parameters that produced the lowest error in the previous step.

The reason why we decided to perform a 3-step parameter tuning is to avoid bias towards the first Blob Detection approach used. We believed that, because the Gamma Transform and the Intensity Mapping are tuned using the DoH approach, it is possible that the parameters of these steps are optimized to be used only in combination with the DoH approach but not with the other two approaches. The third step re-tunes the Gamma Transform and Intensity Mapping to eliminate this bias towards the DoH blob detection approach.

We will show the results of each of the steps in the upcoming sections. The relative error displayed in all figures was computed by averaging the relative error in each of the 120 images used in this process, which was calculated using Equation 5.5.

Step 1: Gamma Transform & Intensity mapping

We wanted to compare the performance of the transformation we chose for the Intensity Mapping step (described in Equation 5.3) with other common transformations. Equation 5.7 describes a Linear Mapping, while Equation 5.8 describes a threshold. We will call Sigmoid Mapping to the transformation we chose for our pipeline to distinguish it from the other two approaches. Figure 5.10 shows a graphical representation of these three transformations.

$$\begin{aligned} f(x) &= k \cdot (x - t) + t \\ 0 &\leq f(x) \leq 255 \end{aligned} \tag{5.7}$$

$$f(x) = \begin{cases} 0, & \text{if } x < threshold \\ \text{x}, & \text{otherwise} \end{cases} \tag{5.8}$$

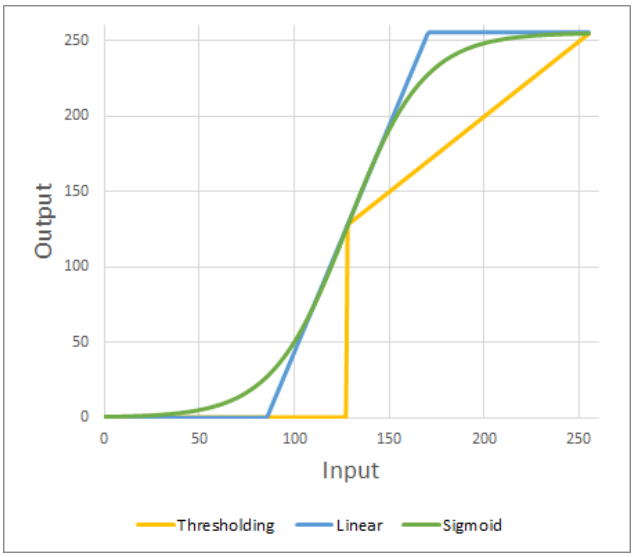

Figure 5.10: Input-Output relationship of the three different Intensity Mapping approaches discussed.

Intensity Mapping: Sigmoid mapping

The lowest relative error obtained was 18.23% using $\gamma = 3, K = 0.5, t = 135$. Table 5.1 summarizes the ranges of values tested for each of the parameters of the Gamma Transform and Sigmoid mapping steps.

Table 5.1: Summary of the range of values tested for each of the parameters of the Gamma Transform and Sigmoid mapping steps.

Parameter	Min	Max
Gamma	1	4
K	0.05	0.5
t	120	160

Intensity Mapping: Linear Mapping

The lowest relative error obtained was 21.24% using $\gamma = 3, K = 3$, $t = 145$. Table 5.2 summarizes the ranges of values tested for each of the parameters of the Gamma Transform and Linear mapping steps.

Table 5.2: Summary of the range of values tested for each of the parameters of the Gamma Transform and Linear mapping steps.

Parameter	Min	Max
Gamma	1	4
K	2	3
t	135	170

Intensity Mapping: Thresholding

The lowest relative error obtained was 19.25% using $\gamma = 3, t = 120$. Table 5.3 summarizes the ranges of values tested for each of the parameters of the Gamma Transform and Thresholding steps.

Table 5.3: Summary of the range of values tested for each of the parameters of the Gamma Transform and Thresholding steps.

Parameter	Min	Max
Gamma	1	4
t	110	140

Gamma Transform & Intensity mapping summary

We have compared the error obtained using different approaches for the Intensity Mapping step. We have observed that the lowest relative error achieved was using the Sigmoid Mapping approach (18.23%). Thresholding has produced the second lowest error (19.25%), being the Linear Mapping the technique that has been the least accurate (21.24% of relative error).

Table 5.4 summarizes the lowest relative error obtained with each of the three approaches and the parameters that produced it.

Table 5.4: Intensity Mapping approaches comparison

Intensity Mapping approach	**Sigmoid**	**Linear**	**Thresholding**
Gamma	3	3	3
K	0.5	3	N/A
t	135	145	120
Relative Error	18.23%	21.24%	19.25%

Step 2: Blob detection

The Blob detection step has 5 parameters for all their approaches. As listed in the *scikit-image* documentation (see [56]):

- **Minimum sigma**, **maximum sigma** and **number of sigma values:** Accepted sigma values for the Gaussian kernel used by the blob detection method.
- **Threshold:** The absolute minimum scale space maxima. Lower threshold values allow detection of weaker blobs.
- **Overlap:** Ratio of overlap between two blobs above which the larger blob absorbs the smaller.

Blob detection: DoG

The DoG approach has **sigma ratio** instead of **number of sigma values** as a parameter, which represents the ratio of the standard deviation of the Gaussian Kernels used. The lowest relative error obtained by the DoG approach was 17.4%. Table 5.5 summarizes the ranges of values tested for each of the DoG blob detector parameters.

Table 5.5: Summary of the range of values tested for each of the parameters of the DoG blob detector.

Parameter	**Min**	**Max**
Max. Sigma	5	12
Min. Sigma	1	6
Sigma ratio	0.5	2.5
Threshold	0.01	2.5
Overlap	0	1

Blob detection: DoH

The Determinant of Hessian approach is the fastest of the three. However, it is unable to detect blobs with a radius lower than 3 pixels accurately. It is because of this that we decided to have a fixed value of **Minimum sigma = 3** for this specific approach. The lowest relative error obtained by the DoH approach was 17.02%. Table 5.6 summarizes the ranges of values tested for each of the DoH blob detector parameters.

Table 5.6: Summary of the range of values tested for each of the parameters of the DoH blob detector.

Parameter	Min	Max
Max. Sigma	4	10
Min. Sigma	3	5
Num. Sigma	1	10
Threshold	0.005	0.06
Overlap	0	1

Blob detection: LoG

The Laplacian of Gaussian is the slowest of the approaches but reported to be the most accurate of the three. The lowest relative error obtained was 23.59%. Table 5.7 summarizes the ranges of values tested for each of the LoG blob detector parameters.

Table 5.7: Summary of the range of values tested for each of the parameters of the LoG blob detector.

Parameter	Min	Max
Max. Sigma	5	9
Min. Sigma	1	5
Num. Sigma	1	10
Threshold	0.005	0.04
Overlap	0	1

Blob detection summary

Table 5.8: Blob detection parameter results on the 2nd tuning step

Blob detection approach	DoH	DoG	LoG
Max. Sigma	6	6	9
Min. Sigma	3	2	4
Num. Sigma/Sigma Ratio	5	1.25	1
Threshold	0.01	0.1	0.03
Overlap	0.5	0.8	0.75
Relative Error	17.02%	17.4%	23.59%

Step 3: Re-tuning Gamma Transform & Intensity Mapping

As mentioned earlier, in this third step we will re-tune the Gamma Transform and Intensity Mapping steps using each of the three Blob Detection approaches. For each of them, we will use the parameters that produced the lowest error during the Blob Detection tuning.

During the Blob Detection parameter tuning, the DoH approach produced the lowest relative error. However, there is a chance that the DoH approach performed better because it is the blob detector that we used during the first step of the parameter tuning.

This third parameter tuning step performs an additional parameter tuning of the Gamma Transform, and the Sigmoid Mapping stages using the three different blob detection approaches with their best combination of parameters. By doing this, we attempt to eliminate the bias towards the DoH blob detection.

Table 5.9 gathers the relative error obtained with each of the three approaches as well as the combination of parameters that produced it. We can observe that the DoH blob detector has given the best results after this third parameter tuning step, followed very closely by the DoG blob detector. Additionally, we can see that the optimal parameters obtained in this third tuning step are identical to those obtained during the first step of the tuning process, which could be an indicator that there was little bias towards the DoH blob detector approach when tuning these parameters for the first time.

Table 5.9: Summary of the 3rd step of parameter tuning

Blob Detection approach	DoH	DoG	LoG
Gamma	3	3	3
K	0.5	0.4	0.3
t	135	135	140
Relative Error	17.02%	17.37%%	20.6%

Parameter tuning summary

We have performed a 3-step parameter tuning on our flower detection pipeline. We started tuning the Gamma Transform and Intensity mapping steps together, using the DoH blob detection approach as the last pipeline's step and compared the performance of different intensity mapping approaches. The second step tunes the parameters of the blob detector step and compares the performance of the three available methods. Finally, we re-tuned the parameters of the Gamma Transform and Intensity mapping steps to eliminate possible bias towards the DoH blob detection approach. We have found that the Sigmoid mapping and the DoH are the Intensity mapping and Blob detection approaches that produced the lowest error (17.02%). However, the DoG blob detector obtained a close relative error (17.4%); therefore both methods could be equally valid for our purposes. Table 5.10 summarizes the parameters that have produced the lowest error.

Table 5.10: Final parameters of the flower detection pipeline after tuning

Pipeline step	Parameter	Value
Gamma Transform	**Gamma**	3
Intensity Mapping	**K**	0.5
DoH Blob Detector	**t**	135
DoH Blob Detector	**Max. Sigma**	6
DoH Blob Detector	**Min. Sigma**	3
DoH Blob Detector	**Num. Sigma/Sigma Ratio**	5
DoH Blob Detector	**Threshold**	0.01
DoH Blob Detector	**Overlap**	0.5
	Relative Error	17.02%

5.5 Flower detection of first days of flowering

After tuning the parameters, we ran images taken during the first days of flowering through our flower detection pipeline. As we discussed in Section 3.1, images have been affected during the early days of flowering. Some cameras did not provide pictures between days 0 and 12 from flowering, while others captured blurry images that can cause our flower detection to be poorly reliable.

We have not found any given time of the day that always produces better images than a different time of the day for flower counting. Therefore, to calculate the number of flowers on a given day and plot, we use the number of flowers detected on all the images captured at that day and plot. Once we have run the images through the flower detection pipeline, we gather the flower counts of all those images and extract the histogram of those flower counts. For example, Figure 5.11 shows the histogram (in blue) of the flower counts obtained from images taken on day 1 from flowering at plot 1109, with a bin width of 5 flowers. If

we calculate the histogram of the flower counts with a bin width of 1 and pass the result through a sliding window, the result is what Figure 5.11 shows in the orange line. At this point, we calculate the number of flowers of a given day as the argument that maximizes the previously described sliding window, which in Figure 5.11 represents the point in the X-axis where the orange curve is maximum. The belief behind this idea is that the majority of the images that agree on the same number of flowers represent the actual number of flowers in that plot on that day. This is also known as *majority voting*. In each day, there is a possibility that our pipeline has many false positives or false negatives in some of the images. This would mean that for some images, the flower count on them is far away from the actual number of flowers that we can see in them. However, it would be very coincidental that our pipeline detected a highly incorrect number of flowers on a subset of images and, at the same time, that all those images agreed on the same flower count. It is much more probable that very few of those images agree on the same flower count. At the same time, we expect that the majority of the images that agree on the same flower count are producing a flower count that is close to the actual value. However, it is also common that these flowers might not agree on the same flower count, but rather on similar flower counts. For example, if 30% of the images agree that there are 99 flowers, another 30% agrees on 100 flowers, and another 30% agrees on 101, with a sliding window of 3 we would obtain that 90% of the images agree on 100 flowers, which is a relatively accurate result.

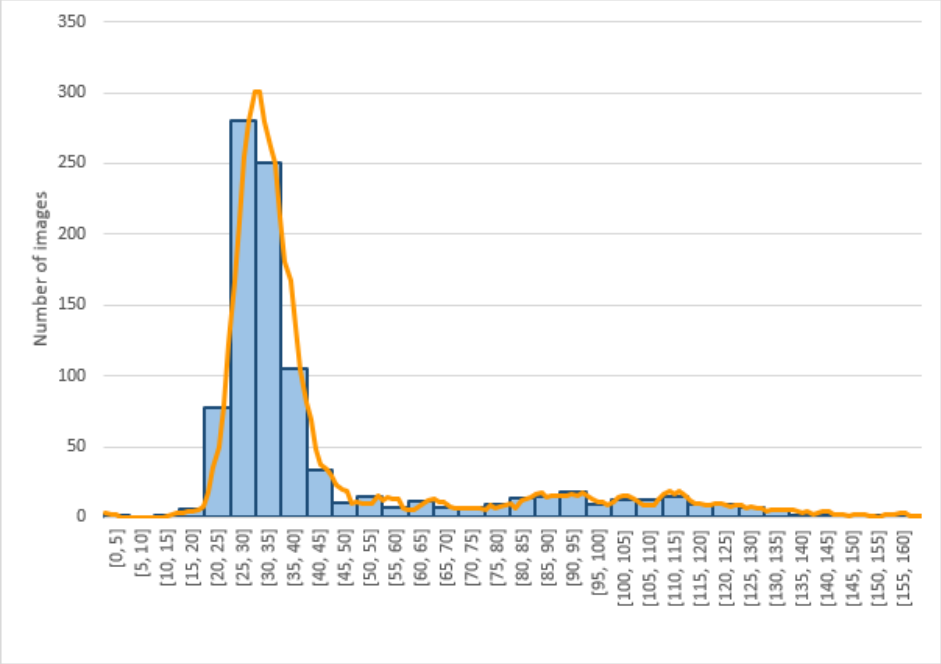

Figure 5.11: Histogram (bin width = 5) of the number of flowers detected on plot 1109 on day 1 from flowering. In orange, the result of passing a 5-unit wide sliding window across the histogram (bin width = 1) of the number of flowers detected on plot 1109 on day 1 from flowering.

The camera situated on plot 1109 had provided sharp images during the first six days of flowering. Therefore we decided to calculate the number of flowers during the first days of flowering on images from this camera. Figure 5.13 shows in blue the flower density of plot 1109 during the first days of flowering obtained from the result of the sliding window method described earlier with a window size of 5. The flower density has been calculated by dividing the number of flowers by the area of the plot ($12m^2$, as discussed at the beginning of Chapter 3). The green line shows the result of a sliding window passed through the number of flowers that were manually counted following the procedure described in Section 5.3, while the red line shows the relative error of this method for each of the days, calculated with the formula described in Section 5.4.1. We see that the error is high on day 0 from flowering (given that our method determines there are 5 flowers, while the manual count determines there are only 3). However, for the following days, the error ranges between roughly 10% to 20%. Figure 5.12 shows the relative error curve for different sliding window sizes, W. As we can see, the relative error is similar for each sliding window size.

We can see that the flower density growth increases notably from day 2, and tends to decrease again after day 4. The flower density is expected to keep growing until approximately day 10 from flowering. However, as the actual number of flowers increase, they will overlap between each other and will make it more difficult to distinguish them.

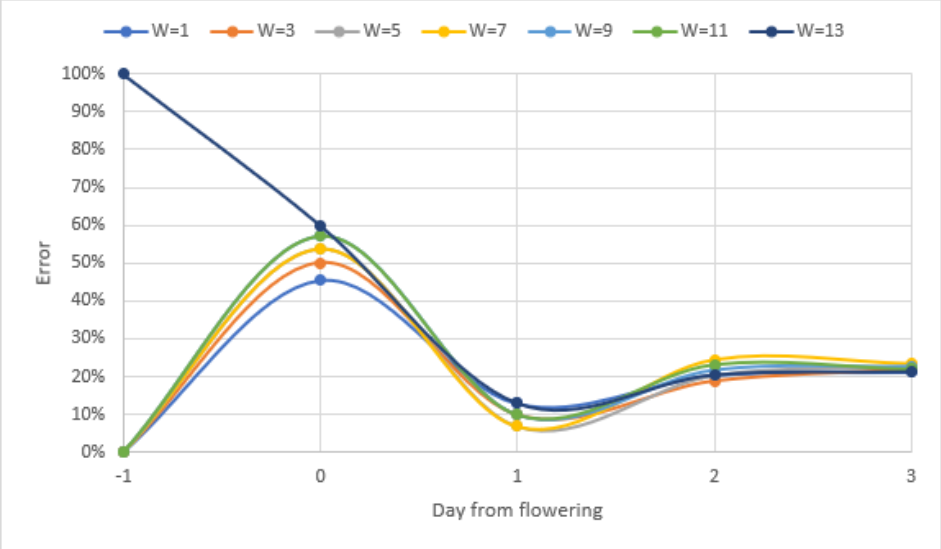

Figure 5.12: Relative error between the calculated number of flowers per day using the sliding window method and the ground truth for different sliding window widths (W).

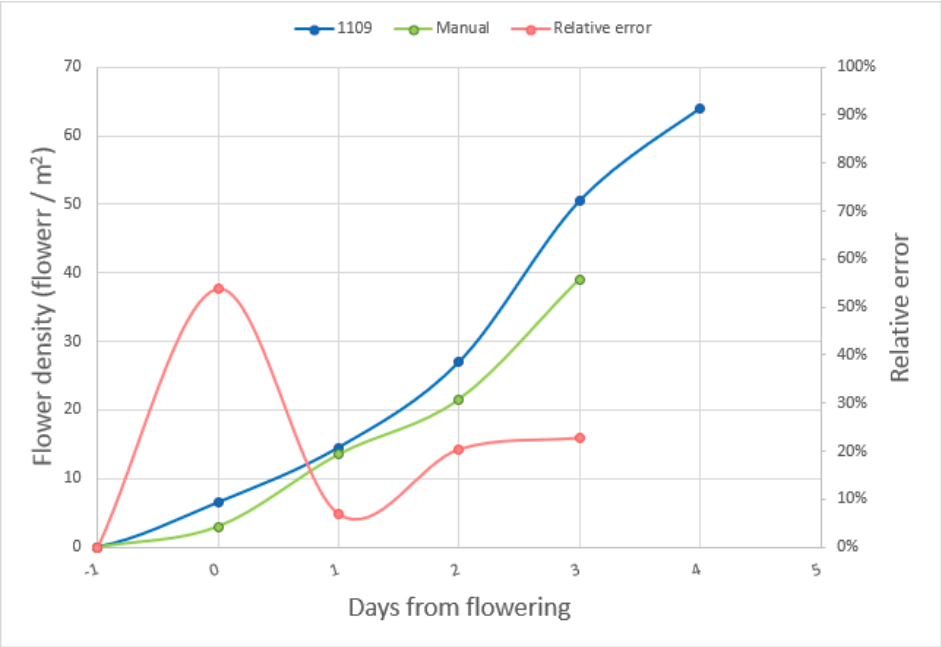

Figure 5.13: Flower density (blue) calculating by the sliding window method described, the flower density calculated from ground truth data (green) and the relative error between the two (red).

5.6 Comparison to previous work

As part of analyzing the performance of our method, we decided to compare our results to similar work. While we have not found related work about counting canola flowers on in-field time-lapse images, we have decided to compare the performance of our method to other image processing approaches designed to detect and count different elements of plants (e.g., fruits, [36, 38] or flowers, [20]) from different species other than canola. The lowest relative error obtained with our method has been 17.02%. Some of the related work achieves high performance, [36], [20], although other designs that work with small flowers present slightly higher error percentage than ours, [38]. Table 5.11 summarizes the comparison between our results and that reflected in other articles about image processing techniques to detect and count plant elements.

Table 5.11: Comparison of relative error and correlation between our method and related work

	Relative error %	Correlation %
Our method	17.02	96.29
[20] (Tangerine flower detection)	17	97
[38] (Orange detection)	6.05	-
[38] (Apple detection)	11.25	-
[38] (Plum detection)	29.8	-
[36] (Orange detection, best)	7.7	99.42
[36] (Orange detection, worst)	9.7	96.88

5.7 Summary

In this chapter, we have presented our flower detection pipeline. For each of the steps of our pipeline, we have tuned their parameters to minimize the relative error. We obtained ground truth by manually counting flowers on a small set of images taken on the first days of flowering.

While we presented an image processing pipeline capable of detecting flowers on time-lapse images, this method is designed to work with images that are not affected by external factors. In combination with this pipeline, it would be optimal to have a step that identifies and treats these affected images, so that the final flower count is not compromised.

Chapter 6

Automatically Detecting Bad Flower Counts

During data acquisition, the time-lapse cameras are set up to capture one image every minute. This means that a single camera generates 960 images from 05:00 to 21:00 on a single day. It is expected that our image processing pipeline detects a similar number of flowers in all of them, as the flower density is not likely to change on the same day significantly.

However, we have obtained a significant disparity of the number of flowers detected at certain moments of the day. Figure 6.1 shows the number of flowers detected between 05:00 and 21:00 during the first days of flowering and the histogram of the flower count (referred to as *HoFC* from now on) from day -1 to 3 from flowering. In the charts, we can see in blue the number of flowers detected by our pipeline at any time of the day and their histograms. In green, we show the manual flower count that we performed in some of the images and their histograms so that we can compare what our pipeline has detected with the flower counts our raters performed. The illustration shows how the flower count significantly varies at arbitrary moments of the day. The number of flowers identified ranges from 0 to over 160 in the early days, and up to 255 when the flower density increases. We can see at the top of Figure 6.1 that on day -1 from flowering or flower counting pipeline detects no flowers for most of the cases. However, during the second half of the day, the number of flowers increases up to almost 350 at 5 PM (the charts in Figure 6.1 are clipped to 150 flowers for displaying purposes). Because we know that it is not possible that the same portion of the plot has 0 flowers during the morning and 350 flowers in the afternoon for the same day, it is evident that there is a subset of images that produce incorrect flower counts. The question that might arise after this description is: "Are there 300 flower or no flowers at all?". Looking at the number of flowers detected along the day and the HoFC at its right, we could assume that the actual number of flowers is zero (given that half of the images say so). However, during the afternoon, for some reason that we do not know at the moment, the images were affected in a way that our flower detection pipeline thinks that there are more flowers than there should be.

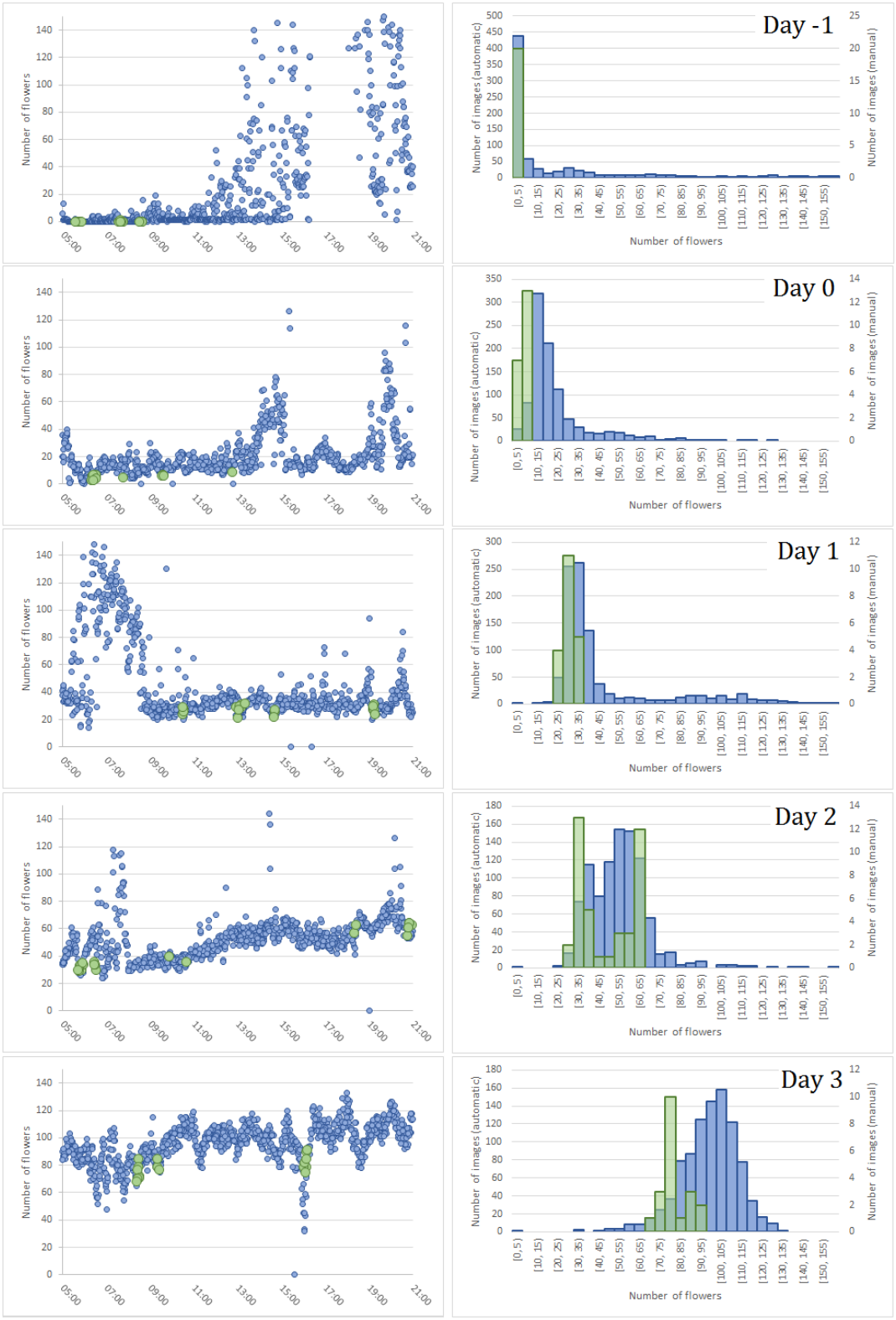

Figure 6.1: Number of flowers counted on the different times of the day (left) and their histogram (right) from day -1 to 3 from flowering. Blue series: number of flowers detected by our pipeline; Green: number of flowers manually counted by the raters (described in Section 5.3).

This introduces the concepts of *good* and *bad* images. An arbitrary image is considered a *good image* if the number of flowers detected on it matches or is relatively close to the actual number of flowers seen in it. On the other hand, a *bad image* is that in which the flower detection pipeline has computed a flower count that significantly differs from the actual number of flowers. Using the description of the flower counts in day -1 from flowering that we described earlier, *good images* are those whose flower count equals or is close to 0, while *bad images* are those images that produced an unusual flower count. It is hard to determine which is the flower count that divides the *good images* from the *bad images*. We said that a *good image* has a flower count that is equal or **close to** the actual flower count. But, how close is close enough? One acceptable way to determine if an image is *good* or *bad* is that, as we reported that our pipeline has a relative error of 17%, a *good image* would have a flower count whose relative error is less or equal than 17%.

We mentioned that we determine whether an image is *good* or *bad* based on how close its flower count is to the actual number of flowers (i.e., the manually annotated flower count) that are seen in the image. However, we will not know what the actual flower count is for every period of time, as we do not have ground truth data for all of them. On the other hand, we do have ground truth for some days, therefore for those days, we can easily identify which images are *good* and which are *bad*. We are interested in finding features that those images identified as *good* have in common, and that separates them from the *bad* images. If we are able to identify these features, then we can look for the same features in images taken on days without any ground truth data, so that we can identify which of those images are *good* or *bad*, and therefore which of those have a reliable flower count.

In this chapter we focus on the analysis of images taken on the early flowering days and the number of flowers counted on them. We aim to learn and understand what the cause of the incorrect flower counts that we mentioned earlier is and discuss methods that could be used to identify common features that *good* images share and that distinguish them from *bad* images. For the analysis presented in this chapter, we have used ground truth extracted following the procedure described in Section 5.3. Our pipeline was set up to detect flowers within the same region in which the users were asked to count flowers (described in Section 5.3) so that we could use that ground truth in this analysis.

6.1 Possible hypothesis and solutions

6.1.1 Using the correlation coefficient between histograms

The flower detection process depends on the content of the CIELab's b channel: as it can be recalled from Chapter 5, a modified version of this channel is passed through the blob detector. We looked at the CIELab's b component of the images taken in some of the days with high flower count disparity. In them, we tried to search for features that could be used to separate the *good images* from the *bad images*. We believed it could be possible that images that produce a correct flower count are similar to each other, but images with an incorrect flower count are not necessarily. As we mentioned earlier at the end of Chapter 5, we believe that

the majority of the images that agree on the same (or similar) number of flowers for a given day represent the actual number of flowers in that day. Similarly, for images whose flower count is incorrect will likely disagree on the number of flowers with other images. At the same time, we believed that images that agree on the same flower counts would have similar CIELab's b component. Therefore their histograms would be similar. On the other hand, two images that do not agree on the same number of flowers will likely have different CIELab's b components, which could mean that their histograms could be different. Given this, we believed that the image that had the lowest flower count error would have a similar CIELab's b histogram than the rest of the images that had a similar flower count. At the same time, the image with the lowest flower count error would have a less similar histogram to that of an image with an incorrect flower count.

To test this theory, we calculated the average of the manual flower counts performed on each day, avg_{manual}. We looked at the image taken on the same day that produced the closest flower count to avg_{manual}. We will call this image *Minimum Error Image (MiEI)*, and its flower count *Minimum Error Flower Count (MiEFC)*. Finally, we calculated the correlation coefficient between *MiEI*'s histogram and the histogram of the rest of the images taken on the same day.

As an example, Figure 6.2 shows the histogram of the *MiEI* and the *Maximum Error Image (MaEI)* from plot 1109 taken on day 1 from flowering. For that plot and day, we obtained $avg_{manual} = 22.297$ and $MiEFC = 22$, while the flower count of *MaEI* was 157. In Figure 6.2, there is a notable difference between the two histograms for high intensities, which correspond to yellow pixels.

We erased the regions of the histograms whose value is zero (in Figure 6.2, $x < 100$ or $x > 180$) and used the remaining values to calculate the correlation coefficient. These regions are not relevant to compute the similarity between the histograms, as their value is always 0 for all the images. Figure 6.3 shows the correlation coefficient between the image that produces the lowest error (i.e., *MiEI*) and the rest of the histograms for days 0 to 5 from flowering. In the case of day 1, we can see that *MiEI*'s histogram has a higher correlation coefficient with images that provide similar flower count. The correlation value obtained decreases as the flower count of the image strays from the manually-obtained flower count. However, some of the lowest correlation values are obtained with images with a good flower count. In general, looking at the charts in Figure 6.3, it is not clear which images could have a good or a bad flower count, which means that our hypothesis that we could use the correlation coefficient between histograms to distinguish between *good* and *bad* images was not correct.

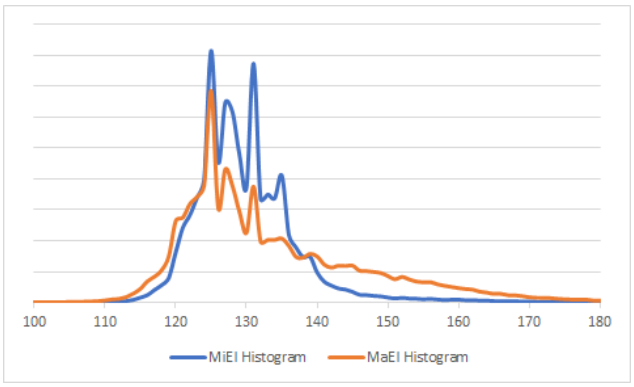

Figure 6.2: Histograms of the CIELab's b channel of *MiEI* and *MaEI*. Zones not showing in the figure have a value of 0

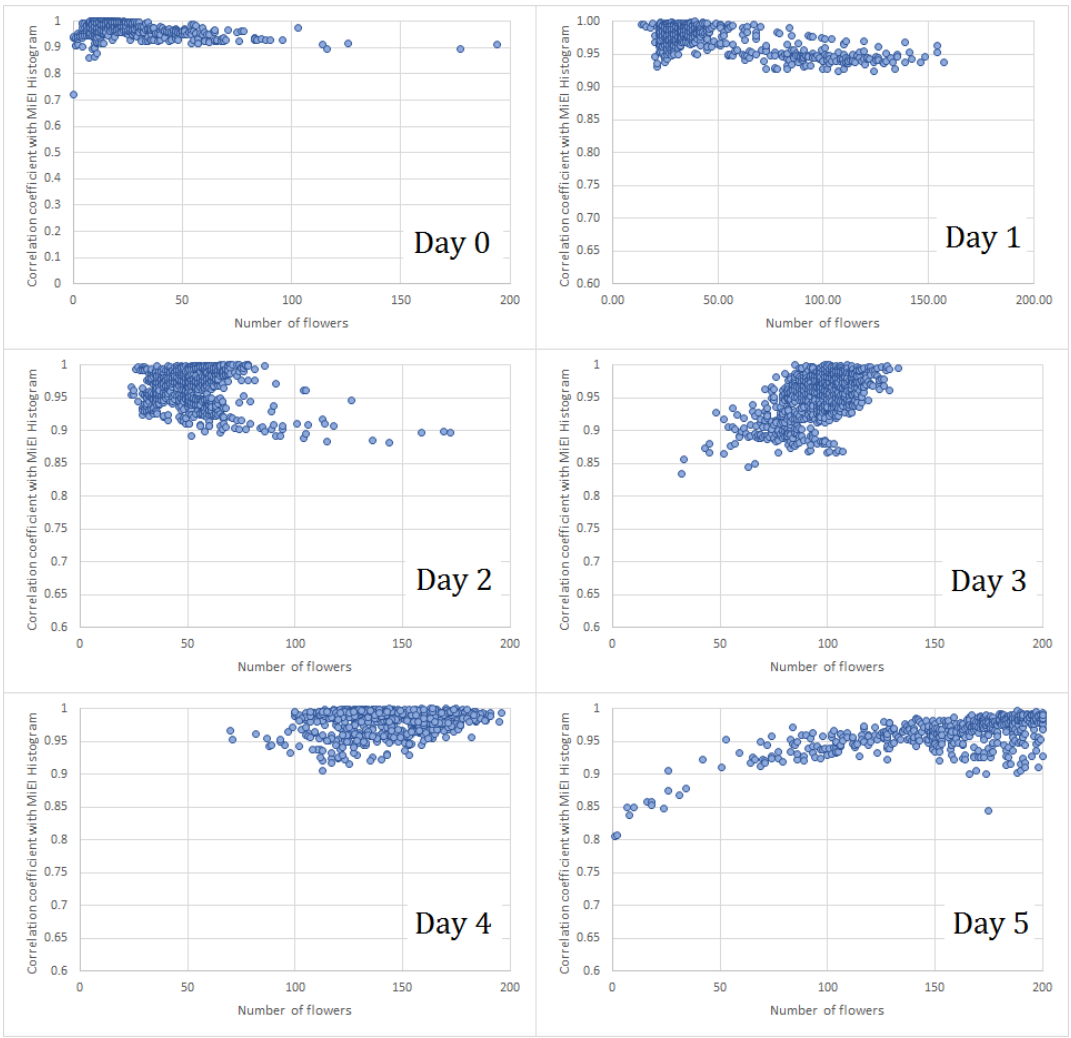

Figure 6.3: Correlation coefficient between the *MiEI*'s histogram and the rest of the image's CIELab's b histograms for each day from day 0 to day 5 from flowering.

6.1.2 Sunlight-based classification

At the start of the development of this thesis, it was believed that the extremely high flower counts in Figure 6.1 were due to the effect that direct sunlight had on the canola leaves. Given that the yellow component increased during sunny periods (i.e., direct lighting), these could make it hard for our pipeline and even the human eye to distinguish the flowers. Section 3.1.1 describes this problem and how it presents in the images.

We started by obtaining ground truth to test said hypothesis. Five people were asked to classify the images based on the following instructions:

- Images shall be classified in two classes based on the scene's lighting: *diffuse sunlight* (class #0) and *direct sunlight* (class #1).

- An image shall be classified as *diffuse sunlight* if little or no direct sunlight is hitting the canola plants. In this case, the absence of direct sunlight makes it easier for the user to distinguish flowers.

- An image shall be classified as *direct sunlight* the sunlight is directly hitting the canola plants(i.e., there is direct lighting). This would cause the user to have difficulties distinguishing flowers from the background.

The users were presented with two examples of *diffuse sunlight* and *direct sunlight* images respectively (see Figure 6.4). The manual classification was done by running a Python script that presented the user with images to be classified. The images used for classification were taken on day 1 from flowering on plot 1109 (960 images in total).

Figure 6.4: Example of diffuse (left) and direct (right) sunlight images presented to the user as a reference for classification.

We observed that the vast majority of the unreliable flower counts were produced by *direct sunlight* images. Figure 6.5 shows the Histogram of Flower Count for day 1 from flowering divided into the two categories. We observe in this image that, while almost all the *diffuse sunlight* images produced good flower counts, a significant amount of the *direct sunlight* images (red histogram) produced good flower counts too. This proves that the hypothesis that the incorrect flower counts are caused by direct sunlight illumination on the canola leaves is wrong.

However, when we looked at the number of flowers detected versus the yellow pixels in the image for all the images from each of the two classes (*diffuse sunlight* and *direct sunlight*), shown in Figure 6.6, we thought that probably our classification was not appropriate. We could think that in Figure 6.6 we see two groups of data points, as it looks like the data form two clusters. However, it could be the case that our *direct sunlight*/*diffuse sunlight* classification was not correct, as we see that some of the *direct sunlight* images are located in the same cluster as the *diffuse sunlight* images. Therefore, there is a possibility that we can classify the images in *good* or *bad* images using the number of yellow pixels and the number of flowers detected, which we evaluate in the next section.

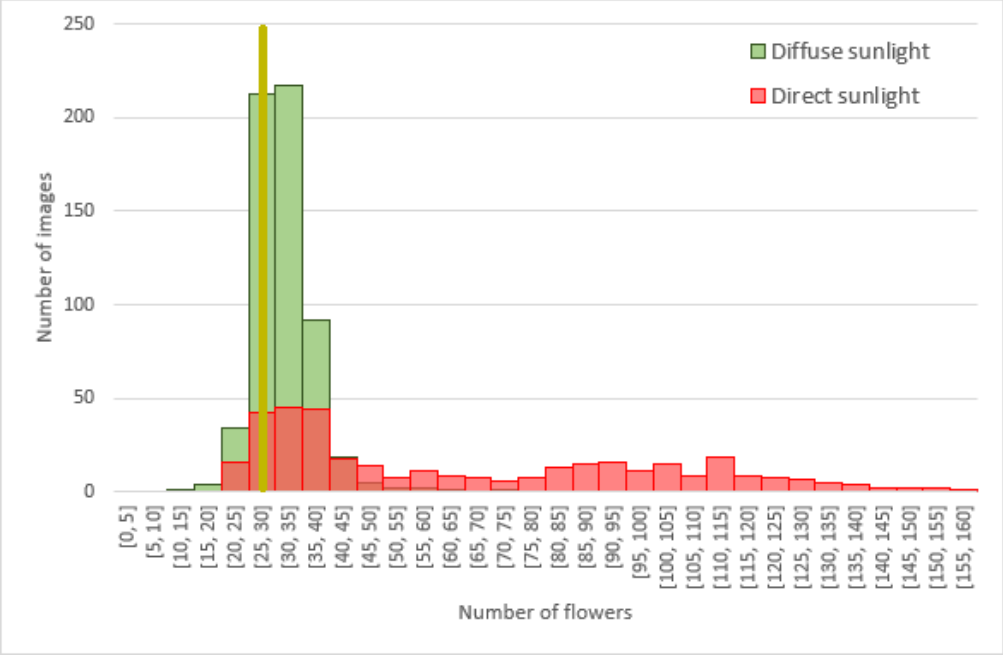

Figure 6.5: HoFC for manually-labeled *diffuse sunlight* (green) and *direct sunlight* (red) images from plot 1109 on day 1 from flowering. A gold vertical line marks the number of flowers in the plot determined by the ground truth.

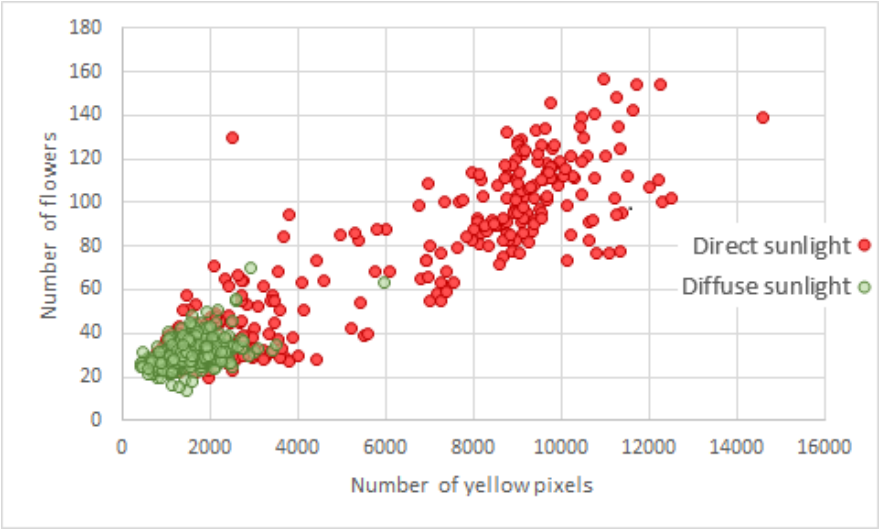

Figure 6.6: Number of flowers vs. yellow pixels, separated by the two classes we described (*diffuse sunlight* (green) and *direct sunlight* (red)), for images from plot 1109 on day 1 from flowering.

6.1.3 Using the number of yellow pixels

The flower detection pipeline looks for yellow blobs in an image. This means that, in general, the number of flowers increases when the total yellow area in an image increases. We believed that we could use the number of yellow pixels to identify which images were more likely to give an incorrect flower count. We hypothesized that, if the number of yellow pixels in an image strongly differed from the majority of the photos, then it would be likely that its flower count strongly differed as well. Computing the number of yellow pixels of an image is computationally less expensive than detecting and counting its flowers. If our hypothesis was correct, we could use the number of yellow pixels to predict which images are likely to produce an incorrect flower count and remove them from further analysis.

We calculated the number of yellow pixels in an image by performing thresholding in its CIELab's b channel (after applying the alignment described in Section 4.3). The threshold value was chosen so that it maximized the correlation coefficient between the number of flowers and the number of yellow pixels. To find the threshold value, we followed this method:

1. Take images from day 0 to 5 from flowering and apply the previously described thresholding using a threshold value T.

2. Calculate the correlation coefficient between the number of flowers detected and the number of yellow pixels per image.

3. Repeat the two previous steps using values $T \in [130, 170]$.

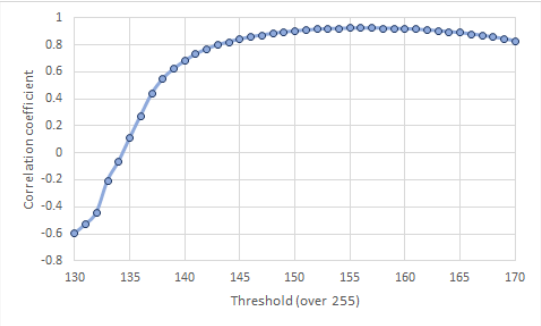

Figure 6.7: Correlation coefficient obtained between the number of yellow pixels and the number of flowers detected on images from day 0 to 5 from flowering on plot 1109 for different yellow threshold values.

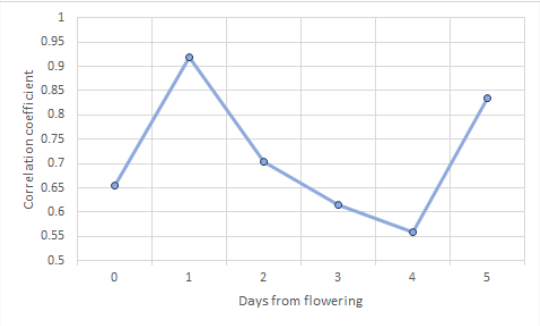

Figure 6.8: Correlation coefficient obtained between the number of yellow pixels and the number of flowers detected on images from each day with a yellow threshold of $T = 157$

Figure 6.7 shows the correlation coefficient values obtained with the different threshold values tested. We obtained a maximum correlation coefficient of 92.27% with $T = 157$, which implies a strong relationship between the number of flowers and number of yellow pixels. However, when we repeated the previously described method only with images taken on the same day and plot, we obtained very different values for every day. As an example, for images taken at plot 1109 on days 1 and 4 from flowering, the correlation coefficient ranged from 92% to 55% respectively. Figure 6.8 shows the correlation values obtained for each day using $T = 157$. This proved that using the described method to compute the number of yellow pixels and that there was not a high and consistent correlation between yellow pixels and flowers.

At the end of the previous section, we briefly introduced the idea of classifying the images into good or bad using the number of flowers detected and the number of yellow pixels in the image, as Figure 6.6 seems to show two clusters of data points based on their position on the chart. To start tackling this possibility, we decided to extract the number of yellow pixels versus the number of flowers detected for every image for various days during the early flowering. Figure 6.9 shows, in the blue series, the yellow pixels vs. the number of flowers detected by our pipeline for all the images from day -1 to 3 from flowering. The green series in each chart represents the number of yellow pixels versus the number of flowers manually counted by our raters. We can see that, for day 1, we can distinguish two clusters of data points. However, for any other day, there is no clear division between two or more clusters of points that could lead to a division between *good* and *bad* images. Given that we have only seen the two-cluster situation for one day, we cannot assume that it will happen for any other day, concluding that we cannot classify images as *good* or *bad* based on the relationship between the number of yellow pixels and number of flowers detected.

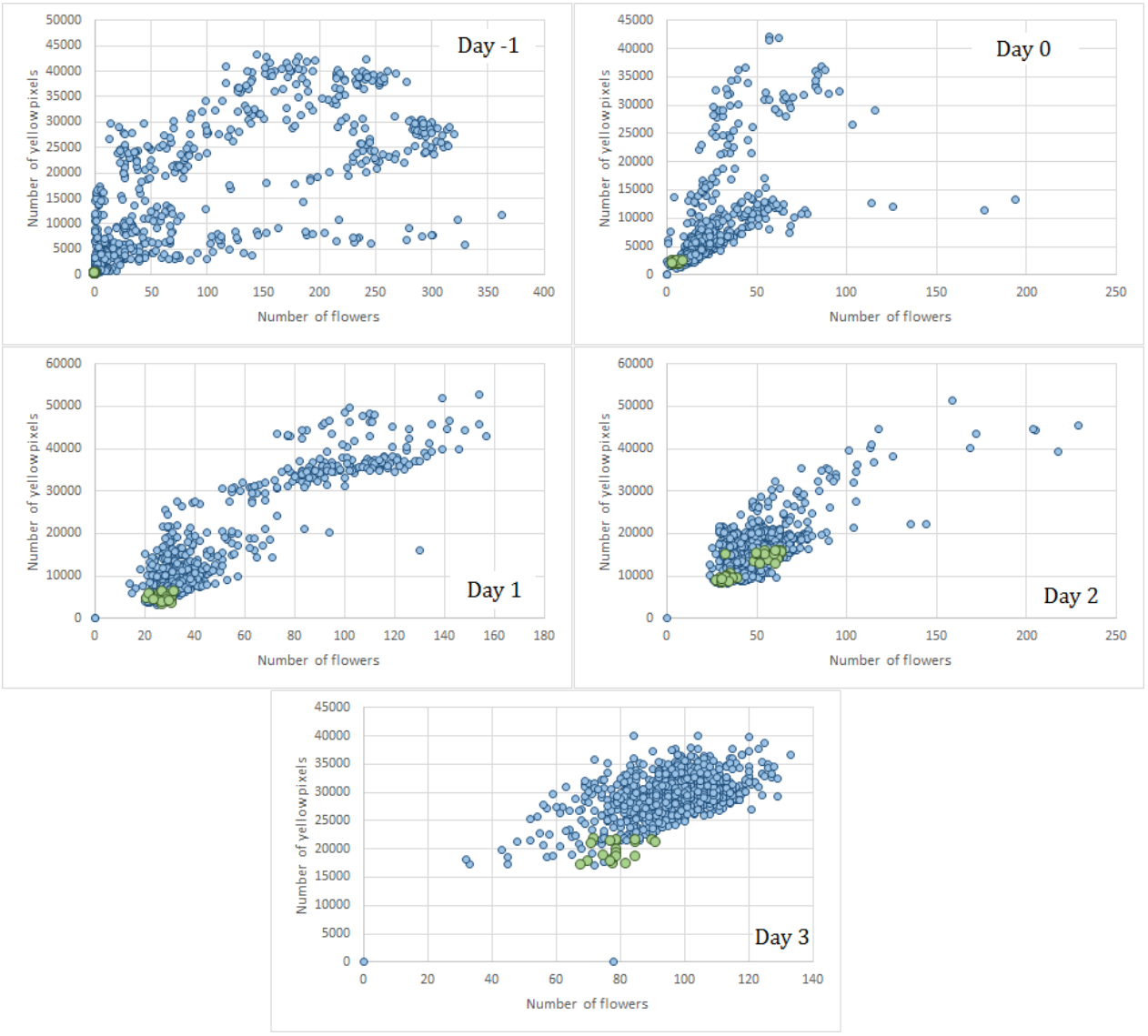

Figure 6.9: Yellow pixels vs. number of flowers automatically detected (blue series) and manually counted (green series) in all images from day -1 to 3 from flowering.

6.1.4 Using histogram of flower count (HoFC)

One of the approaches that were discussed is to make use of the *HoFC* of the images of each day. After performing flower count for the first time, the actual number of flowers in the plot is unknown. Taking Figure 6.1 as an example, the number of flowers detected for day 1 ranges from 0 to 160, which implies that some images are not producing reliable flower counts. However, we can assume that:

- Images whose flower count falls in those zones where the histogram is more compact are very likely to have a reliableflower count. In the case of Figure 6.1, this is the case of the images whose flower counts fall roughly between 20 and 40 for day 1.

- Images whose flower count falls in those zones where the histogram is more spread are very likely to have an unreliable flower count. In the case of Figure 6.1, this is the case of the images whose flower counts is higher than 40 for day 1.

Using this observation, we can define a feature named **Compactness of the HoFC**. This feature is an intuition we initially had of *"how compact the HoFC is around a given value"*. For example, in Figure 6.1, for day 1 from flowering, we could think that the histogram looks more compact around the red line (i.e., around $X = 27$), while in the other areas it seems spread. We did not operationalize this concept at the beginning. Nevertheless, we found that the sliding window procedure that we described in Section 5.5 is an operationalization of our intuition that describes the Compactness that we talked about.

Using the sliding window should have two main advantages:

- It is independent of the number of flowers the images present. The sliding window over the HoFC will be higher on those flower count values where the HoFC is higher, independently of what those flower counts could be.

- It is independent of the ratio between the images that give a reliable flower count versus those that produce an unreliable result. Should all the images produced an unreliable result, the HoFC would be evenly distributed, and thus the sliding window result would be almost constant. On the other hand, if all the images produce a reliable flower count, it is expected that the HoFC is more compact, therefore having a high peak in the sliding window result.

To evaluate the performance of our theory about using the HoFC, we have calculated the compactness of the histogram by calculating a sliding window of width 5 over the histograms plotted in Figure 6.1. Figure 6.10 shows the result of a sliding window over the histograms of the number of flowers calculated by our pipeline (in blue) and performed manually by the raters (in green) for the days from -1 to 1 from flowering. The peak of the green series indicates the point where most of the raters have agreed on the number of flowers for a given day (with an error of +/- 5 flowers). If the compactness of the HoFC was a good indicator of whether an image has a reliable flower count or not, we would expect the compactness of the histogram of the automatic flower counts (i.e., the blue series) to be high where the compactness of the histogram of the manual flower counts (i.e., the green series) is high too.

In Figure 6.10 we can see that for days -1 and 2, the peak for the sliding window results from both automatic, and manual flower counts match with a small error (1 flower), which indicates that our method and the raters have agreed on the same number of flowers for those days. However, day 0 from flowering, there is a striking distance between the peaks (8 flowers). This indicates a disagreement between our pipeline and the raters, which is indicating that using the compactness of the HoFC might not be a good method to identify *good* and *bad* images.

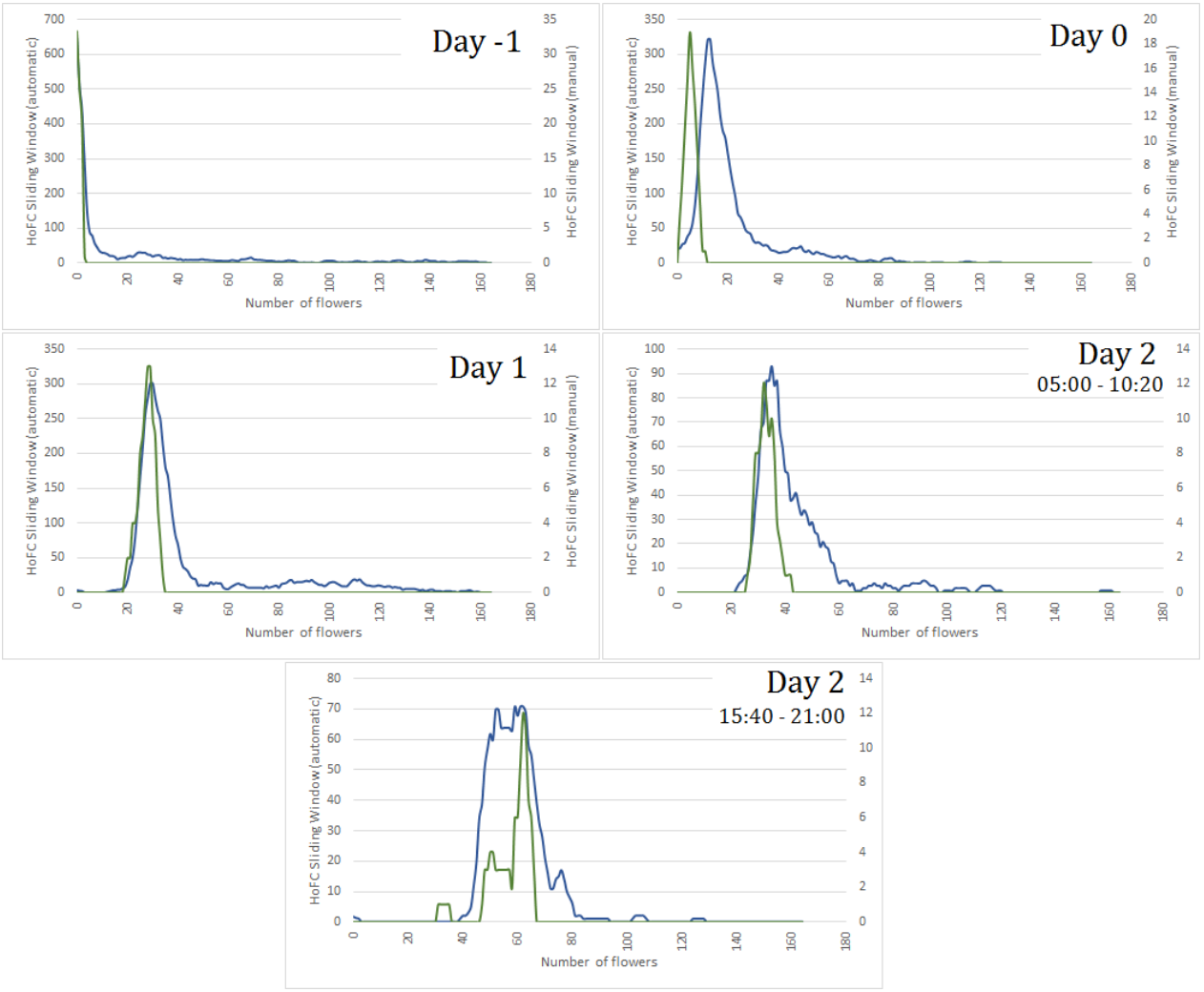

Figure 6.10: Sliding window over the HoFC using automatic flower counts (blue series) and manual flower counts (green series) from days -1 to 2 flowering. For day 2 we show 2 of the 3 periods in which we divided the day (sliding window between 10:20 and 15:40 not showing due to lack of manual flower counts for that period)

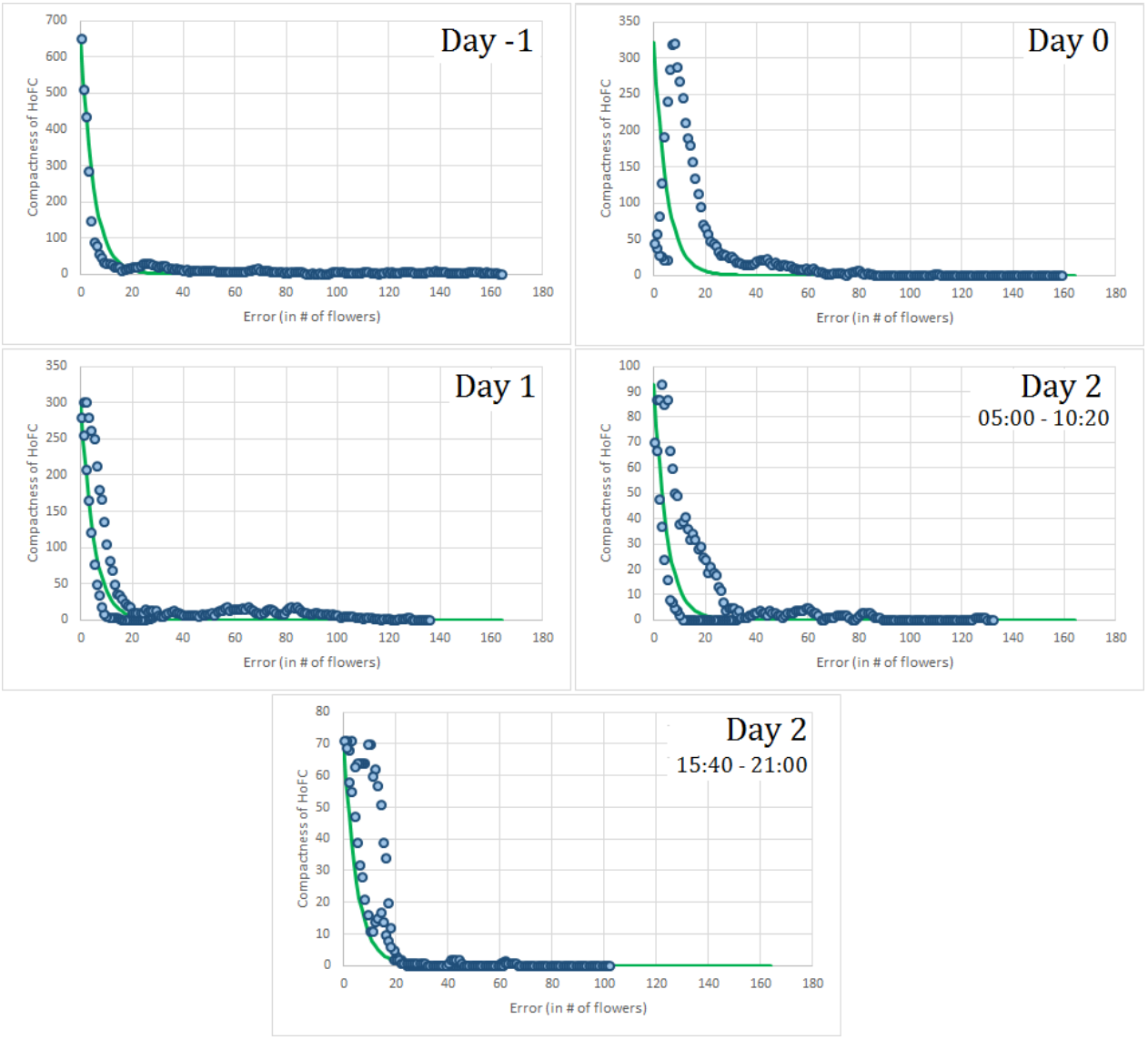

Figure 6.11: Error (calculated as flower count difference) between automatic and manual flower counts (X axis) against compactness of the HoFC (Y axis) for every image from day -1 to 2 from flowering. The green series shows an intuition of what the chart should ideally look like, included for displaying purposes only.

To get a sense of how well the compactness of the HoFC works for flagging good/bad images, we want to calculate the compactness of the HoFC vs the error in the flower count for each image. We assumed that, given that the peak of the green series indicates the number of flowers on which our raters have agreed for a given day, we would interpret that value as the actual number of flowers for a given day. This means that, for day -1, 0 and 1, the actual number of flowers in the field was 0, 6 and 28 respectively. We calculate the error on the flower count of an image as the difference between the actual number of flowers and the flower count calculated by our pipeline. In Figure 6.11 we plotted the error of an image with a flower count X

against the compactness of the histogram, Y, at the position X, from day -1 to 2 from flowering. This is basically the blue series of the charts in Figure 6.10 shifted horizontally so that the peak of the green series falls on $x = 0$, which gives us a sense of whether high compactness relates to a reliable flower count. To give an idea of what we expected in these charts, we included the green series as a reference, which represents an intuition of what the data should look like in an ideal case. This reference series was included for displaying purposes only and has not been used in the analysis of this method. We consider the green series to be a representation of an ideal case because, for high values of compactness, the error is low, while the error increases as the compactness decreases. We also expect a saturation region, which represents the flat region of the green series, as the compactness is almost constant and close to zero when the error is above a certain value.

As we expected, for days -1 and 1, the error is low for those flower counts with high compactness. In the case of day 0, this is true as well; however, there are some flower counts with even lower error and low compactness (i.e., the points near to the coordinates origin), which indicates not an excellent performance of this method. However, the compactness relates to flower count reliability for days -1 and 1.

We analyzed the data from day 2 from flowering, and we have to take a different approach here. If we see the HoFC for day 2 in Figure 6.1, we can predict that the compactness of HoFC will not have a good result either for this day. However, in that same figure, if we look closer at the number of flowers over time for day 2, it looks like the number of flowers grows along the day, which is something that we can expect, as the flower growth rate is high during the first days of flowering. Even if we look at the histogram of the ground truth in Figure 6.1 for day 2, we can see two local maxima, which correspond to the number of flowers manually counted at the beginning and the end of the day respectively.

If the number of flowers increases through the day, then we must divide the day into different portions and evaluate them separately. We can divide day 2 into 3 portions between 05:00 and 21:00: the first goes from 05:00 to 10:20, the second goes from 10:21 to 15:40, while the third period corresponds to the remaining time slot. Figure 6.10 also shows the compactness of the HoFC for the first and the third periods (given that, as we can see in Figure 6.1, there is no ground truth for day 2 between 10:20 and 15:40). In this case, the compactness looks better, and the maxima for both green (manual flower count) and blue (automatic flower count) series seem to be closer to each other than in the HoFC shown in Figure 6.1. We calculated the error vs compactness similarly to what we did for days -1, 0 and 1 from flowering. The results show, again, that images with high compactness have a small flower count error, which supports the hypothesis that the compactness of HoFC can be used to identify which images have a reliable flower count and which have not.

We attempted to perform the same analysis for the data from day 3 from flowering. Figure 6.12 shows the compactness of HoFC using manual (green) and automatic (blue) flower counts for day 3 from flowering. We can see that there is a considerable error between the automatic and manual flower count. However, as we have mentioned before in this thesis, day 3 contains a significant amount of flowers and not only can it be difficult for our raters to manually count flowers due to the high flower overlapping, but also the flower

detection pipeline might not be as precise as it could be. Due to the images' quality, it can be hard for both raters and the flower detection pipeline to perfectly distinguish how many flowers they see in a cluster of flowers. In case of day 3, an image with a flower count with high compactness (i.e., where the sliding window is high) is not necessarily an image with low error. However, as we said, it is difficult to validate our method for day 3 on from flowering, as the quality of the images does not allow to distinguish flowers when they are overlapping with each other perfectly.

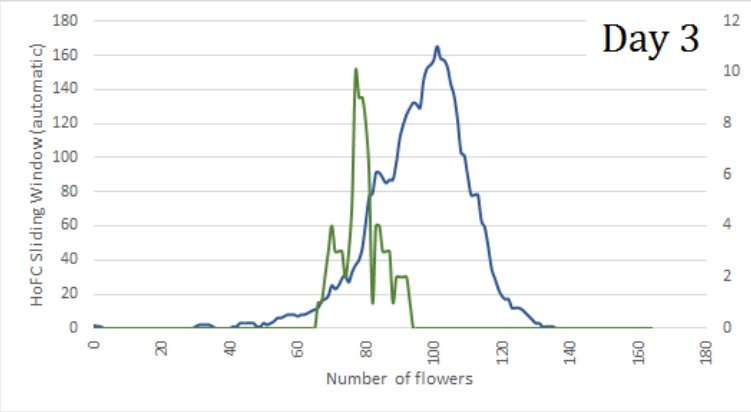

Figure 6.12: Sliding window over the HoFC using automatic flower counts (blue series) and manual flower counts (green series) for day 3 from flowering

6.2 Summary

We have discovered that our image processing pipeline detects an unusually higher number of flowers on a significant amount of images for a given day. It is important to filter these pictures out, as they are not producing reliable flower count values. We proposed and discussed different hypotheses and methods that can be applied to achieve this filtering to provide a basis for future work.

Some of the hypotheses have seemed to be right under certain circumstances, but it is not assured that they will be true for every set of images. Among all the methods proposed, using the *Compactness* of the Histogram of Flower Count (*HoFC*) has proven to be useful to identify which images have provided an incorrect flower count after the fact. While the rest of the methods rely on having a relationship between the number of flowers and other features obtained prior flower detection (e.g., number of yellow pixels or the histogram shape), needs only the number of flowers detected in each image, calculating the HoFC and its compactness from the flower counts.

Chapter 7

Future Work

During the development of this method, we have only used images from 2016 season. However, new images were taken during the summer of 2017. Images from 16 cameras can be retrieved from 2017's database, all of which were recording different traits of canola, while only six time-lapse cameras were installed in 2016's season. 5 of the 16 cameras for 2017 season were Brinno TLC200 Pro, while the other 11 are custom cameras that use a Raspberry Pi programmed to take a picture roughly every 5 minutes, while 2016's cameras were configured to capture a new image every minute. These new cameras were powered using solar panels, removing the need to take them down to change the batteries. Additionally, by using a Raspberry Pi, we could retrieve the images by connecting a USB cable without moving the cameras, which takes care of possible changes in the camera's orientation. Figure 7.1 shows an example of one of the images generated by one of the 2017's season cameras.

Figure 7.1: Example of a time-lapse image taken during the 2017's season.

To extract the information provided in this section, we performed a visual inspection of the images that are available in 2017's season database. Figures 7.2 and 7.3 show the data available that can be found in the 2017's database. For Figure 7.3 we divided the images in *OK*, *Bad* or *Undetermined* based on the quality of the images, which depends on blurriness, colour or lens filters that the cameras used (e.g., camera from plot 1122 uses a NIR lens filter). In both figures, the flowering date is marked with a blue square. This date has been calculated by visually inspecting the images and searching for the first day that flowers can be seen in them. While the flowering date that we calculated for these charts can be used as a reference, it might not match the actual flowering date (e.g., it can be the case that flowers appeared one day before the flowering date). We can see that most of the cameras have taken images during the early flowering and some of them provided images during peak flowering. However, some of them (e.g., the cameras in plots 1130 or 1213) do not provide images during late flowering. Additionally, the camera placed in plot 1146 did not provide any images during flowering; therefore that camera cannot be used.

Figure 7.2: 2017's season data availability of each plot across time based on the quality of the images of the 11 custom cameras.

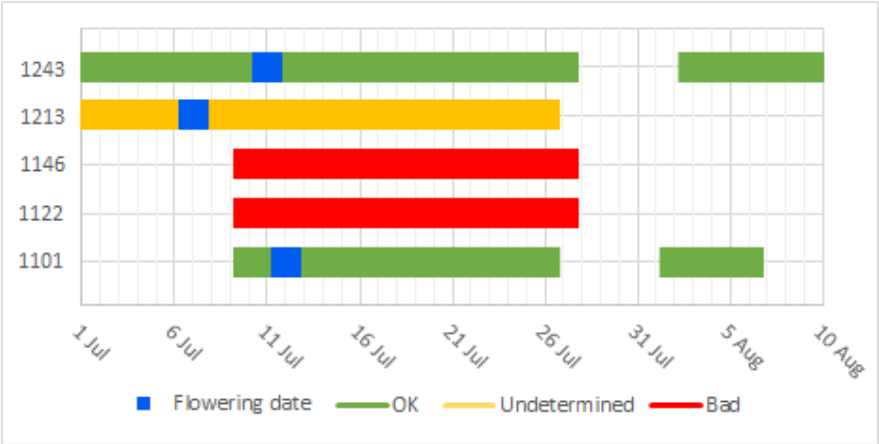

Figure 7.3: 2017's season data availability of each plot across time based on the quality of the images of the 5 Brinno cameras.

Despite using a different model of camera, we will still face some problems mentioned on this thesis, such as dealing with blurry images (e.g., cameras from lots 1243 or 1105), see Figure 7.4 for example. Additionally, we found that some images still present a sunlight intensity problem. In some cases, the sunlight is so intense that identifying flowers can be very hard. Figure 7.5 shows an example of an image affected by this phenomenon.

In the case of the Brinno cameras, we found that the colour setting on cameras placed in plots 1122 and 1213 does not seem to be same as for the rest of the cameras. 1122's camera uses a NIR filter, which makes the images to have a high yellow intensity. This will most likely strongly affect the results of our pipeline, meaning that we cannot use the images from this camera with our pipeline design.

Figure 7.4: Difference between a blurry image (left) and a sharp image (right) taken by the 2017's season cameras.

Figure 7.5: Difference between a direct sunlight image (left) and a diffuse sunlight image (right) taken by the 2017's season cameras.

7.1 Suggestions on how to use 2017's data

After visually inspecting the images from 2017's season, we believe that the cameras placed on plots 1143 and 1242 will be the most useful to improve our method. They all provided sharp images during most of the flowering season. Additionally, the camera on plot 1142 provided sharp images during early flowering, which will be very useful as well. Images from all these plots could be used to extract ground truth and use it to finer tune the flower detection pipeline.

On the other hand, images taken by cameras from plots 1105, 1215 or 1243 might not be beneficial, as they generated blurry images, despite having worked during the entire flowering season. Additionally, the camera from plot 1146 did not provide images during any of the flowering season days, thus being of no use for flower detection.

Some of the suggested methods and approaches to use 2017's include:

1. Testing whether using images taken with a different camera model affects the performance of our pipeline. It is possible that our pipeline, which is tuned to work on images extracted from the Brinno TLC200Pro cameras, might not perform as good as expected with images taken by a different camera, as the colour balances are not the same. For example, canola leaves on Brinno cameras look usually darker than images extracted with 2017's cameras. This might make a difference in the CIELab colour space, thus forcing us to re-tune the flower detection pipeline.

2. Ground truth acquisition. As mentioned earlier, images taken on plots 1143, 1242 and 1142 might be a good source of ground truth. The flowers look sharp on them, and they provide images during the early flowering days, which are the easiest for a user on which to manually count flowers.

7.2 Suggestions facing future seasons

In this thesis, not only did we present and discussed our flower counting method but also we gave an overview of the challenges that had to be faced during 2016's data acquisition term. One of our purposes when going over those challenges is to learn from the unforeseen problems that we encountered and try to anticipate them in future seasons.

As we discussed in Section 3.1, there were two main problems that we had to face after extracting the data:

- Missing data: There were long periods of time for which we did not have any images. This limits the amount of data that we can use to improve and validate our method. Our belief is that dead batteries and memory cards running out of space were the main reasons for this problem, which can be overcome by estimating *a priori* how long the batteries will last and how long will the camera take to fill up the memory cards, and thus going to the field to either change the batteries or empty out the memory cards.

- Blurry images: There were large amounts of images that were blurry due to a bad camera focus, which also limits the amount of data that can be used to develop our method.

The camera model used for 2016's season, the Brinno TLC200 Pro time-lapse camera is easy to set up, but after working with its images, it might not be the best choice for this project. Based on our opinion, a different time-lapse camera model should be used for future seasons, which allows an easier image extraction from the memory cards and an appropriate focus adjustment. Using a different time-lapse camera could prevent these two problems, providing a much larger amount of high-quality images to be used for this method.

7.3 Summary

While only data from 2016 has been used for the development of this thesis, there is much more data extracted on the summer of 2017. This can serve as a new source of ground truth, refining the flower detection pipeline and verify the methods that we presented here. It will be interesting to know whether our design works for images taken using a different camera model and if the pipeline needs to be readjusted or modified. Through an in-depth analysis of 2017's images, we aim to develop a more robust method to detect canola flowers using time-lapse images.

Chapter 8

Conclusion

In this thesis, we have presented an automated method for canola flower detection and counting on images acquired during the first days of canola flowering. We have performed a parameter tuning to gather the optimal combination of parameters that reduce the relative error of our method. Using images from the first days of flowering, we have used our flower detection pipeline to detect flowers on pictures from one of the plots taken in the Summer of 2016.

In the early chapters, we have reviewed the main limitations and challenges that we had to face during the development of this thesis. One of the main problems that we had to face was data availability. Most of the images from the first days of flowering were missing, making it difficult to use the available pictures to validate our method. However, a plan to use the images extracted on Summer of 2017 is provided, along with an overview of the data availability. While we did not use photos from 2017 during the development of this thesis, we believe that the higher data availability will help us improve our method. Additionally, we found a significant amount of blurry images that make it difficult for our image processing pipeline and raters to count the number of flowers. As we discussed in Chapters 3 and 7, we believe that using a different camera model will help solve the missing data and blurry images problem. Also, we believed that the direct sunlight illumination on the canola leaves was the cause of many incorrect flower counts; however, we saw in the latest chapters that the sunlight was not the cause.

We believe our method can be considered somewhat robust, given that we have used a small dataset for validation and that we did not have access to large portions of high-quality data due to missing or blurry images, as we have reviewed earlier. While we could personally consider our method reliable for the early flowering season when properly tuned for the images used, we do believe that its reliability can be significantly improved with an appropriate dataset (i.e., a complete dataset of high-quality images).

We identified a series of initial steps through which the images should go through before any further processing. We would need to execute these just one time after image extraction. We developed a simple method to extract the timestamp from the images, which was printed in the image itself, instead of being embedded in the image's metadata. We also developed a tool with which a user could define a plot's boundaries. Assuming that the cameras do not move during the season, these boundaries could be valid for all the images taken from the same camera throughout the season. Finally, we observed that the histogram of the CIELab's b channel was shifted between pictures produced by the same camera within the same day.

We developed a simple algorithm to compute how much these histograms were misaligned so that it could be corrected during the following image processing steps.

We developed an image processing pipeline that manipulates the images individually and detects the number of flowers present within a specified region within the plot boundaries. This pipeline first modifies the intensities of the RGB's Red and Green components of each pixel, then transforms the intensity CIELab's b component with the aim of highlighting flowers over the background. Finally, an OpenCV's implementation of the Determinant of Hessian (DoH) blob detector is used to localize and count the flowers present. Chapter 5 also shows how we tuned the parameters of this pipeline by using ground truth obtained through performing manual flower counts over a subset of images. Additionally, we validated the steps of the pipeline by comparing their performance with other similar approaches, such as the Determinant of Gaussian (DoG) or Laplacian of Gaussian (LoG) blob detectors.

After running different images through our pipeline, we observed that some of them produced a highly inaccurate flower count value. Chapter 6 attempts to address this problem. We offer a discussion on different hypotheses as to why we could be having these incorrect flower counts. We provided deep data analysis and different procedures that could be used to identify *a priori* which images are prone to produce an incorrect flower count.

Finally, this thesis provides a brief overview of the data extracted in the summer of 2017. Without deep analyzing the images, the data availability seems undoubtedly higher than in 2016. However, the steps of our flower detection pipeline were designed and tuned to use images taken with a Brinno TLC200 Pro; therefore, it is possible that re-tuning or even additional steps are needed to process these images. Nevertheless, 2017's dataset contains images from 5 Brinno TLC200 Pro time-lapse cameras; therefore it is likely that re-tuning or modifying the pipeline will not be needed to process these images. However, data availability for images from the Brinno cameras does not look as good as for the rest.

In our research, we take advantage of having high temporal resolution images, captured through time-lapse cameras, that will allow us to monitor the evolution of canola flowers over shorts periods of time. As opposed to other work that relies on obtaining data under controlled environments, our method handles in-field images, whose use in research is not as extended as indoors phenotyping, thus leading to an evolution in HTP.

References

[1] Canola council of canada 2016 annual report. Brochure.

[2] Canola council of canada website. https://www.canolacouncil.org/.

[3] Computing intraclass correlations (icc) as estimates of interrater reliability in spss. http://neoacademic.com/2011/11/16/computing-intraclass-correlations-icc-as-estimates-of-interrater-reliability-in-spss/.

[4] The daily - principal field crop areas, june 2017. http://www.statcan.gc.ca/daily-quotidien/170629/dq170629c-eng.htm.

[5] A Abinaya and S.M.M. Roomi. Jasmine flower segmentation: A superpixel based approach. *Communication and Electronics Systems (ICCES), International Conference on*, 2016.

[6] A.D Aggelopoulou et al. Yield prediction in apple orchards based on image processing. *Precision Agriculture*, 12:448–456, 2011.

[7] Bronte S. Bergasa L.M. Alcantarilla, P.F. Fog detection system based on computer vision techniques. *IEEE International Conference on Intelligent Transportation Systems*, pages 1–6, 2009.

[8] P Basu et al. A novel image-analysis technique for kinematic study of growth and curvature. *Plant Physiology*, 145:305–316, 2007.

[9] H. Bay et al. Surf: Speeded up robust features. *Lecture notes in Computer Science*, 3951, 2006.

[10] R Benmehaia, D Khedidja, and M.E.M. Bentchikou. Estimation of the flower buttons per inflorescences of grapevine (vitis vinifera l.) by image auto-assessment processing. *African Journal of Agricultural Research*, 11:3203–3209, 2016.

[11] D.C Boyes et al. Growth stage-based phenotypic analysis of arabidopsis: A model for high throughput functional genomics in plants. *The Plant Cell Online*, 13:1499–1510, 2001.

[12] A Campillo et al. Time-lapse analysis of stem-cell divisions in the arabidopsis thaliana root meristem. *The Plant Journal*, 48:619–627, 2006.

[13] L Chaerle et al. Robotized time-lapse imaging to assess in-planta uptake of phenylurea herbicides and their microbial degradation. *Physiologia Plantarum*, 118:613–619, 2003.

[14] J Chaki and R. Parekh. Plant leaf recognition using shape based features and neural network classifiers. *(IJACSA) International Journal of Advanced Computer Science and Applications*, 2:41–47, 2011.

[15] Haralick R.M. Chen, S. Recursive erosion, dilation, opening, and closing transforms. *IEEE Transactions on Image Processing*, 4:335–345, 1995.

[16] X Chen et al. Automated segmentation, classification and tracking of cancer cell nuclei in time-lapse microscopy. *IEEE Transactions on Biomedical Engineering*, 53:762–766, 2006.

[17] C.M Costa and S. Yang. Counting pollen grains using readily available, free image processing and analysis software. *Annals of Botany*, 104:1005–1010, 2009.

[18] M.P. Diago et al. Assessment of flower number per inflorescence in grapevine by image analysis under field conditions. *Society of Checmical Industry*, 94:1981–1987, 2013.

[19] U. Dorj et al. A comparative study on tangerine detection, counting and yield estimation algorithm. *International Journal of Security and Its Applications*, 7:405–412, 2013.

[20] U Dorj and D.U. Imaan. A new method for tangerine tree flower recognition. *Communications in Computer and Information Science*, 353:49–56, 2012.

[21] A.E. Fonseca et al. Aplication of fluorescence microscopy and image analysis for quantifying dynamics of maize pollen shed. *Annals of Botany*, 42:2201–2206, 2002.

[22] A. French et al. High-throughput quantification of root growth using a novel image-analysis tool. *Plant Physiology*, 150:17841795, 2009.

[23] Nguyen T.Q. Gibson, K.B. Fast single image fog removal using the adaptive wiener filter. *IEEE International Conference on Image Processing*, pages 714–718, 2013.

[24] C. Granier et al. In-field variability detection and spatial yield modeling for corn using digital aerial imaging. *Transactions of the American Society of Agricultural Engineers*, 42:1911–1920, 1999.

[25] C. Granier et al. Phenopsis, an automated platform for reproducible phenotyping of plant responses to soil water deficit in arabidopsis thaliana permitted the identification of an accession with low sensitivity to soil water deficit. *New Phytologist*, 169:623–635, 2006.

[26] R.S Harmsen and N.J.J.P. Koenderink. Multi-target tracking for flower counting using adaptive motion models. *Computers and Electronics in Agriculture*, 65:7–18, 2009.

[27] A. Hartmann et al. Htpheno: An image analysis pipeline for high-throughput plant phenotyping. *BMC Bioinformatics*, 12(148), 2011.

[28] K. Huang. Application of artificial neural network for detecting phalaenopsis seedling diseases using color and texture features. *Computers and Electronics in Agriculture*, 57:3–11, 2007.

[29] A. S Iyer-Pascuzzi, O. Symonova, et al. Imaging and analysis platform for automatic phenotyping and trait ranking of plant root systems. *Plant Physiology*, 152(3):1148–1157, 2010.

[30] H.G Jones, R. Serraj, et al. Thermal infrared imaging of crop canopies for the remote diagnosis and quantification of plant responses to water stress in the field. *Functional Plant Biology*, 36(11):978–989, 2009.

[31] J Kim et al. Mobile-based flower recognition system. *Intelligent Information Technology Application, 2009. IITA 2009. Third International Symposium one*, pages 580–583, 2009.

[32] H. O. L. Laue. *Automated Detection of Canola/Rapeseed Cultivation from Space.* 2014.

[33] D.G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the International Conference on Computer Vision*, 2:1150–1157, 1999.

[34] Y Ma et al. An counting and segmentation method for blood cell image with logical and morphological feature of cell. *Chinese Journal of Electronics*, 11, 2002.

[35] A.B. Mabrouk et al. Image flower recognition based on a new method for color feature extraction. *IEEE*, 2015.

[36] Z. Malik et al. Detection and counting of on-tree citrus fruit for crop yield estimation. *International Journal of Advanced Computer Science and Applications*, 7:519–523, 2016.

[37] S Mao-jun et al. A new method for blood cell image segmentation and counting based on pcnn and autowave. *Communications, Control and Signal Processing, 2008. ISCCSP 2008. 3rd International Symposium on*, pages 6–9, 2008.

[38] N. Meena et al. Development of detection, counting and yield estimation algorithm for agricultural products. *International Journal of Engineering Research & Technology (IJERT)*, 3:590–595, 2014.

[39] M. Minervini et al. Image analysis: The new bottleneck in plant phenotyping [applications corner]. *IEEE Signal Processing Magazine*, 32(4):126–131, 2015.

[40] S. P. Mohanty et al. Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7:14–19, 2016.

[41] Tripathi A.K Mukhopadhyay, S. Single image fog removal using bilateral filter. *IEEE International Conference on Signal Processing, Computing and Control*, pages 1–6, 2012.

[42] M. L. Neilsen et al. Extending watershed segmentation algorithms for high-throughput phenotyping. *International Conference on Computer Applications in Industry and Engineering*, 2016.

[43] B Neumann et al. High-throughput rnai screening by time-lapse imaging of live human cells. *Nature Methods*, 3:385–390, 2006.

[44] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on systems, man, and cybernetics*, 9:62–66, 1979.

[45] A. Paproki et al. A novel mesh processing based technique for 3d plant analysis. *BMC Plant Biology*, 2012.

[46] S. Paulus et al. Surface feature based classification of plant organs from 3d laserscanned point clouds for plant phenotyping. *BMC Bioinformatics*, 2013.

[47] A.B. Payne et al. Estimation of mango crop yield using image analysis segmentation method. *Computers and Electronics in Agriculture*, 91:57–64, 2013.

[48] M. P. Pound et al. Deep learning for multi-task plant phenotyping. *IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 2055–2063, 2017.

[49] M. P. Pound et al. Deep machine learning provides state-of-the-art performance in image-based plant phenotyping. *GigaScience*, 6:1–10, 2017.

[50] C. Prnpanomchai et al. Leaf and flower recognition system (e-botanist). *IACSIT International Journal of Engineering and Technology*, 3:347–351, 2011.

[51] Reid D. M. Qaderi, M. M. Growth and physiological responses of canola (brassica napus) to uv-b and co2 under controlled environment conditions. *Physiologia Plantarum*, 125:247–259, 2005.

[52] A. K Reyes et al. Fine-tuning deep convolutional networks for plant recognition. In *CLEF*, 2015.

[53] S. Sankaran and L. R. Khot. Low-altitude, high-resolution aerial imaging systems for row and field crop phenotyping: A review. *European Journal of Agronomy*, 70:112–123, 2015.

[54] R.S Sarkate et al. Application of computer vision and color image segmentation for yield prediction precision. *Information Systems and Computer Networks (ISCON), 2013 International Conference on*, pages 9–13, 2013.

[55] scikit image. Blob detection - skimage v0.14dev docs. `http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_blob.html`.

[56] scikit image. Module: feature - skimage v0.14dev docs. `http://http://scikit-image.org/docs/dev/api/skimage.feature.html`.

[57] A Sharma et al. Overlapped flowers yield detection using computer-based interface. *Perspectives in Science*, 8:25–27, 2016.

[58] S Sladojevic et al. Deep neural networks based recognition of plant diseases by leaf image classification. *Computational Intelligence and Neuroscience*, 2016, 2016.

[59] V. S Sundar and J. Bagyamani. Flower counting in yield approximation using digital image processing techniques. *International Journal of Advance Research In Science And Engineering*, 4:97–106, 2015.

[60] T. O Tayo and D. G. Morgan. Quantitative analysis of the growth, development and distribution of flowers and pods in oil seed rape (brassica napus l.). *The Journal of Agricultural Science*, 85(1):103–110, 1975.

[61] S. van der Walt et al. scikit-image: image processing in python. *PeerJ*, 2014.

[62] Hoo Zhou Yang. *Automated fruit and flower counting using digital image analysis*. 2015.

[63] W. Yang et al. Combining high-throughput phenotyping and genome-wide association studies to reveal natural genetic variation in rice. *Nature Communications*, 2014.

[64] M. Zaman-Allah, O. Vergara, et al. Unmanned aerial platform-based multi-spectral imaging for field phenotyping of maize. *Plant methods*, 2016.

[65] R Zhou et al. Using colour features of cv. 'gala' apple fruits in an orchard in image processing to predict yield. *Precision Agriculture*, 13:568–580, 2012.