Technology
Arts Sciences
TH Köln

University of Cologne

Universität zu Köln & Technische Hochschule Köln | POL InnoBioDiv

2021/22

Automated detection of simulated pests using FarmBot

Protocol - Group 3

Febin Chollapra, Ruth Neeßen, Adriana González & David Günter

Abstract

Early prevention and detection of plant-pathogens plays a vital role in modern agriculture to minimize yield-loss and is one of the main factors to improve efficient land-use. But the common, broad application of pesticides to entire fields is highly controversial in regard to environmental- and health-threads. In this experiment we used the FarmBot-system as the basis for the image-recognition of a simulated pest for minimal application of pesticides. Here we demonstrate the capabilities and thresholds of the "basic" FarmBot-system and compare them with the accuracy needed to sufficiently detect pathogens on *Zea mays* and *Pisum sativum* in early stages.

Content

List of Figures:

List of Acronyms by appearance:

GPS - Global positioning system
CNC - Computerized numerical control
TH - Technische Hochschule
sp. - not further specified species
v - version
px - pixels
GUI - Graphical user interface
BGR - Blue green red (openCV image color-space)
RGB - Red green blue (usual image color-space)
HSV - Hue saturation value (image color-space)
API - Application programming interface
Scikit - SciPy Toolkit
OpenCV - Open Source Computer Vision Library

Introduction

In modern agriculture three factors become more and more relevant. The first being digitalization and atomization taking more and more hold in agriculture. From GPS-controlled (Stoll & Kutzbach, 2000) harvesters, through sensors monitoring the field the entire day to drones imaging the acre (Stehr, 2015), modern agriculture is adopting technology to maximize yield while reducing costs. One example of how high-tech is reaching agriculture is the FarmBot (FarmBot, 2022). An open source agricultural CNC developed at the California Polytechnic State University in 2011, bringing high-tech agriculture into people's gardens, monitoring and watering the field. Another huge factor is pests, sickening or even killing the plants, a problem that is getting even more severe in common monoculture acres. Globalization and climate change play an important role here, while new pests are passively imported as invasive species (Dean R. Paini, 2016), climate change often widens the timeframe in which pests can flourish on plants. Therefore the third factor is the application of pesticides which is mostly done in a wide and unspecific way to the entire field, resulting in problematically high amounts often polluting the groundwater (Hallberg, 1989) and harming the natural rhizo-biom in the soil (Borruso, Mimmo, & al., 2021). In cases of early pest detection, the amounts of pesticides used can be strongly reduced by a more localized application in the field (Berenstein, Shahar, & Shapiro, 2010). In this project we tried to help mitigate the problem of pollution through pesticides by using the FarmBot and image recognition for pest detection. The overall goal is to develop a software tool which could be implemented into the programmed FarmBot routine. The FarmBot system used here, at the TH-Cologne and University of Cologne, is already equipped with sensors for soil-moisture and generally navigates to plants which get under the set threshold for soil-moisture to accurately water them. The idea is that in addition to watering, a picture is taken with the camera available for the FarmBot to be processed with openCV by first masking the images, removing anything from the image outside a green border to further only detect discolorations on the plant-area. In the next step three different protocols were run to analyze for the presence of pests. One detecting yellow, one detecting red and a third that detects everything but green. The coordinates of the pixels detected in the image can then be translated back into coordinates for the FarmBot for a potential reaction to the pest such as treatment with a pesticide. For example a syringe-pump could be attached to the FarmBot,

treating the plants with minimal amounts of pesticides at the exact spot the pest was detected and thus significantly reducing the use of pesticides.

Materials & Methods

Biological

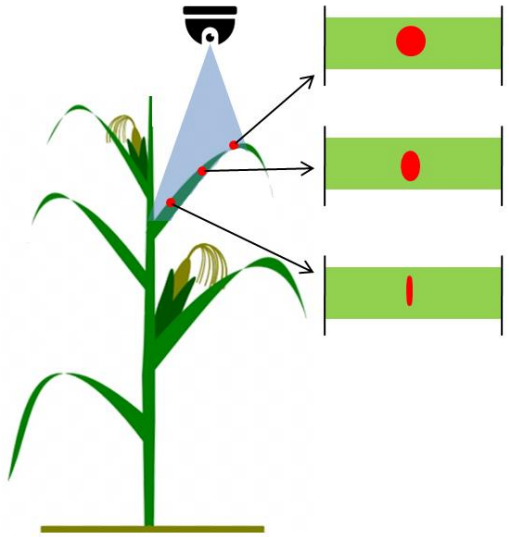


Fig. 1: Illustration of a mays plant and how a top-down image of a red circle changes over the course of a leaf.

The choice of plants for this experiment fell on *Zea mays* and *Pisum sativum* for two reasons. The first being, that both plants are highly relevant in agriculture and therefore suited our thesis of creating an application for modern agriculture. Secondly, due to their very different growth. *Pisum sativum* is, in the early stage we used it, more flat and has broad leafs, that are orthogonal to the Z-Axis of the FarmBot, whereas *Zea mays* gets very tall, very fast with leafs starting almost parallel at the shoot and bending towards being orthogonal at the tip. Thus *Zea mays* was used for taking pictures of "pests'' at different angles to further explore the limits of a top-down optical pest detection as illustrated in Fig. 1.

For the experiment, we used 12 *Zea mays* and 12 *Pisum sativum* plants in order to take pictures of both single plants and an entire tray of plants.

Since the plants were grown in a general greenhouse at the University of Cologne, the option of using real pests was immediately ruled out because the risk of infecting surrounding plants of other experiments was way too high. Thus a "fictional" pest was simulated by adding red

and yellow stickers to the plants. The yellow stickers were chosen to most realistically simulate leaf-spot of pests, while the red stickers were mainly for testing the “prototype”-algorithm due to the significant color-difference on a green background.

Here we want to emphasize the term of a “fictional” pest, due to the difficulty of simulating a pests dynamics. The problem is in the fact that an insect-pest (i.e. potato-beetle, *Leptinotarsa decemlineata*) is very mobile and implicates an infection of the entire field, whereas fungal (i. e. Leafrust, *Puccinia sp.*) or bacterial (i.e. *Xanthomonas campestris*) pests have mostly reached a critical abundance when leaf-spots appear. Thus we simulate a “fictional” pest and focus more on the technical capabilities and potential application with the FarmBot.

During the growth of the plants, they got infested with thrips - an insect pathogen creating yellowish to white spots on the leafs. This accidental infection came in very handy, since now we had real leaf-spots the algorithm could be tested with as an addition to the stickers.

Technical

The images were taken manually with the FarmBot WebApp (my.farm.bot) by moving the FarmBot to the measured coordinates of the tray the plants were in. Pictures were then also taken via the WebApp. The Farmbot at the TH cologne uses the Lindy 43300 Webcam which offers a resolution of 1920 x 1080px. At the university of cologne, the farmbot uses the Genesis v1.5 / Express v1.0 borescope camera with a picture resolution of 640 x 480px.

For programming we used Python 3.8.10 and additionally the libraries openCV 4.5.5.62, numpy 1.22.1, scikit-image 0.19.1, matplotlib 3.5.1.

To be able to exchange the code faster, we used Jupyter Notebook version 0.9.0.

I. Taking the image

The images were taken using Farmbot GUI and Express v1.0 borescope camera and Lindy 43300 Webcam. From there we downloaded the images.

II. Image processing

For the detection of pests on the plant, we used color recognition. All non-green spots on the plant are identified as pests here.

However, there are two problems that require pre-processing of the input image before the actual pest detection can be started:

1. Color correction: different environments and times of day produce different light, which in turn affects the colors of the image
2. Delete background: Everywhere in the image are non-green color areas. Of course, those should be excluded from the detection.

1. Color Correction

We implemented a histogram matching for color correction. Since the matching of color maps is not yet implemented, figure 2 was taken as the reference image here (see discussion).

We used the match_histograms(image, reference, channel_axis=-1) (Scikit Histogram, 2022) method provided by the python library scikit-image. This method creates a histogram of intensities for each channel of an image (see figure 2) and changes the pixel values of the input image whose histogram will be closer to the reference images histogram.

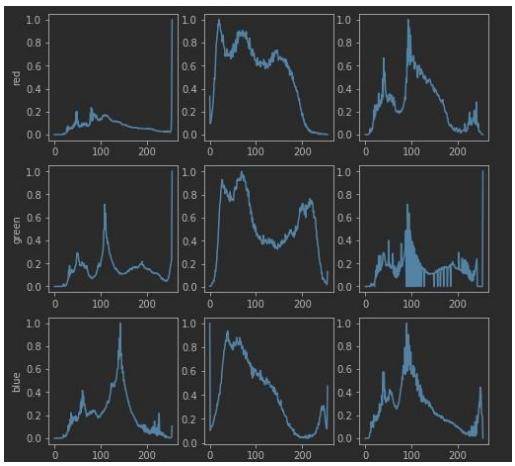


Fig 2: Histograms of each channel for input image, reference image and resulting image.

Delete background

To ensure that spots are detected only on the plants, we had to remove the background. Since openCV automatically loads images in BGR color space, we first converted it to RGB space and saved it in a variable for further use (in the following we refer to it as image1, see figure

3). For color detection we additionally stored the image in HSV space in a variable, we refer to it as image2. The green mask (mask1) was created with the inRange(image, lower, upper) method (OpenCV inRange, 2022) from the OpenCV framework. This method passed the lower and upper bounds of green in HSV color space and applied to image2. From mask1 and image1, we used the bitwise_and(image1, image2, mask) method (OpenCV bitwise_and, 2022) to create an image with only green present (image3). We passed the image1for the first two parameters and mask1 as the third. This determines which area of the image is to be processed by the method. The resulting image contains only green colors (image4), as shown in figure 4.

Fig 3: Input Image (image1) in RGB Space

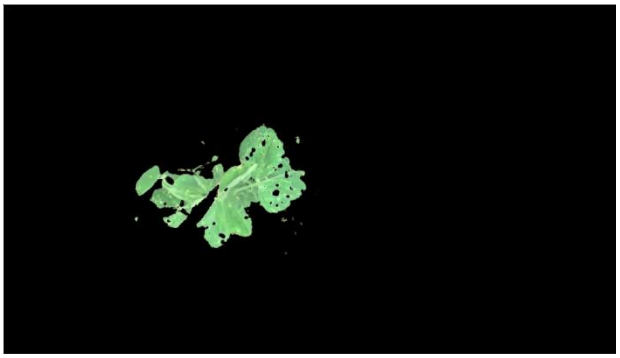


Fig 4: Input image with green Mask (image4)

We could not yet use this image for recognition, because elements of a different color on the plant were also deleted. However, since we were now able to determine the outer boundaries of the plants, we used this image to create a mask that removes everything outside the plant from the original image.

To include as many non-green points on the plant but close to the edge as possible, we created a 5 x 5 structure element with getStructuringElement(shape, size) (OpenCV getStructuringElement, 2022) to widen the plant edge.

Then we splitted the resulting image (image4) into the three RGB channels using openCV's split(image) method (OpenCV split, 2022). The green channel of the image and our structure element was passed to the morphologyEx(image, morphType, structuringElement, iterations) method (OpenCV morphologyEx, 2022), which generated image (image5) of the plant with a wider edge.

Now, to find the plant contours on image5, we used openCV's findContours(image, mode, method) method (OpenCV findContours, 2022). We used this to just find outlines, but it is possible to search for all contours with this method.

Since with time algae or moss had collected on the soil of the plant pots, the contours were also for returning overgrown perlite and clods of soil. For this reason, we first reworked the contours. But findContours returns contours as tuples, which cannot be edited. So we had to convert the tuples of contours to a list. The iteration over the list returns an array with 4 reference points for a spanned area. If the area was smaller than a certain value (in our case 140.0), it was removed from the list.

From the found contours we could then create a mask again (mask2). Unlike the green mask before, we did not create this mask using color values. The filled contours have different red values, and red occupies two distinct areas in the HSV color space. In this case, it was easier to convert the image with the contours to grayscale, and create mask2 using openCV's threshold(input, thresholdValue, maxValue, type) method (OpenCV threshold, 2022). Mask2 contains only values above a set threshold. To create an image, which only contains the masked area, we used the bitwise_and operation once more with our image1 and the mask2. The result is the image with the background removed (figure 5), referred to as image6.

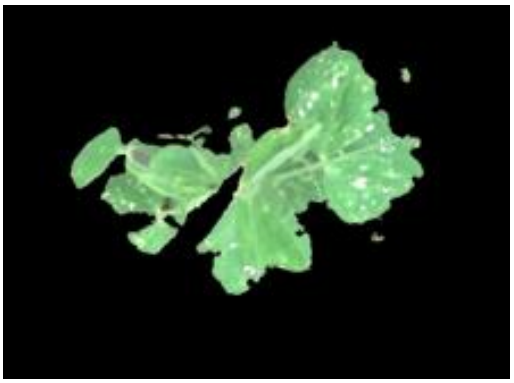


Figure 5: The original image with removed background, image6

III. Pest detection

To detect the spots on the plants, we converted image6 back to an HSV image (image7) and set a color range for image7, from which we create the mask (mask3). This color range should cover the colors we are looking for on the plants. From mask3 and image6 we generated again via bitwise_and a new image (image8), which shows only the image areas of mask3, see figure 6

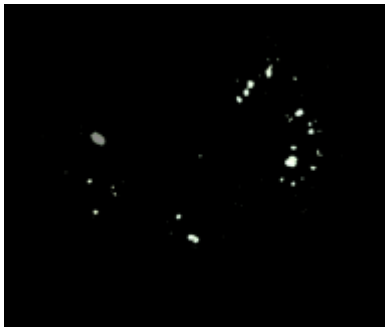


Figure 6: Spots found pea plant (image8).

Results

Input:

In the first approach, an OpenCV-script was used to detect red color in the RGB-spectrum using live-imaging. The results were stable but lacked the biological relevance, since the first indicators for an infection are leaf-spots which are mostly yellow. Furthermore the life-imaging prevented further data being collected for a hypothetical integration of machine-learning-software. To mitigate these problems, the code was altered to use a picture taken by the FarmBot which was first preprocessed and then the pest detection was started.

Light and colors:

The first step in image processing should be color correction. Since we initially did not have access to the Farmbot and could not take any pictures ourselves, we could not use color maps for the color correction. Therefore, we performed the correction with histogram matching using an image from the internet as a reference (figure 7) to be able to implement the algorithm. But here color charts should be used for histogram matching, see discussion.

The pictures we used as input images were taken with the Genesis v1.5 / Express v1.0 camera of a Farmbot at the greenhouse and showed pots of Arabidopsis thaliana. As can be seen in figure 8, the light is quite yellow and there are large light reflections on the plant.

The results from this color correction are shown in figure 8 and figure 9.

Figure 7: Arabidopsis thaliana tray (Shutterstock.com) used as reference picture

Figure 8: Input image before color correction

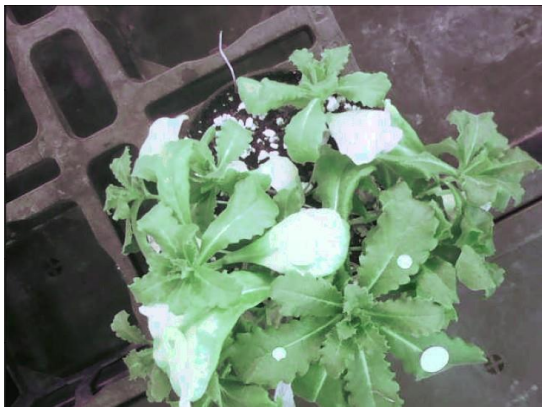


Figure 9: input image after color correction

Later we got access to the Farmbot at the TH Cologne. We were finally able to take pictures of the plants we had actually planned for, namely maize and peas. The pictures were taken by the camera Lindy 43300 Webcam and showed a much better quality. There were no major light reflections on the leaves here and the light was much more neutral. In figure 10 we see a picture of a pea plant, taken by the Lindy 43300 Webcam at the TH Cologne. The picture 11 shows a picture of Arabidopsis thaliana taken by the Express v1.0 borescope camera at the university's greenhouse.

Fig. 10: Pea plant taken by the Lindy 43300 Webcam

Fig. 11: A. thaliana taken by the Express v1.0 borescope camera

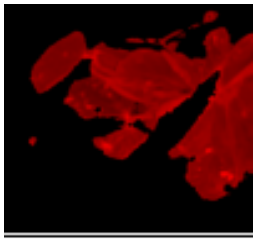
The fact that these large light reflections were no longer present with the new camera made things easier. We had previously been looking for a way to satisfactorily separate them from spots on the plant. Neither was this possible, nor were the outlines of the plant cleanly recognized, let alone spots on or near the light reflections.

Plant detection

As described, we detected the plants by their color, and used a contouring method that finds and stores the outer boundaries of the plant.

Spots at the edge of the plant are excluded as non-green when the first mask (mask1) is created and can therefore no longer be detected by the algorithm. In order to include spots close to the edge in mask2, we extended the plant edge a bit before we determined the contours.

Figure 12 and 13 show the difference in using a structure element to widen the plant edge and not using it, by showing the plants contours.



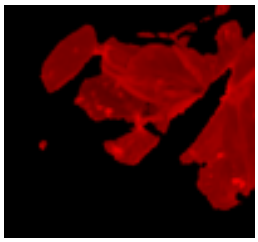


Figure 12: Outline without structuring element

Figure 13: Outline with structuring element

Of course, the outline of the plant cannot be enlarged arbitrarily, as this would result in spots outside the plant being detected again. This leads to the fact that spots which lie on the edge are excluded during the recognition of the plant (mask1) and therefore cannot be recognized later.

Another problem is that holes between the leaves showing the soil and parts of other backgrounds are not excluded. This is due to the fact that we currently use only outlines for contouring. Figure 14 shows how this affects detection.

As already mentioned, after some time algae grows on the soil around the perlite. Since these are greenish, they are picked up in Mask1 and their contours are recognized.

With the correction for algae/moss growth on the soil and perlite, we were able to remove greenish perlite and soil from our contours, as figure 15 shows.

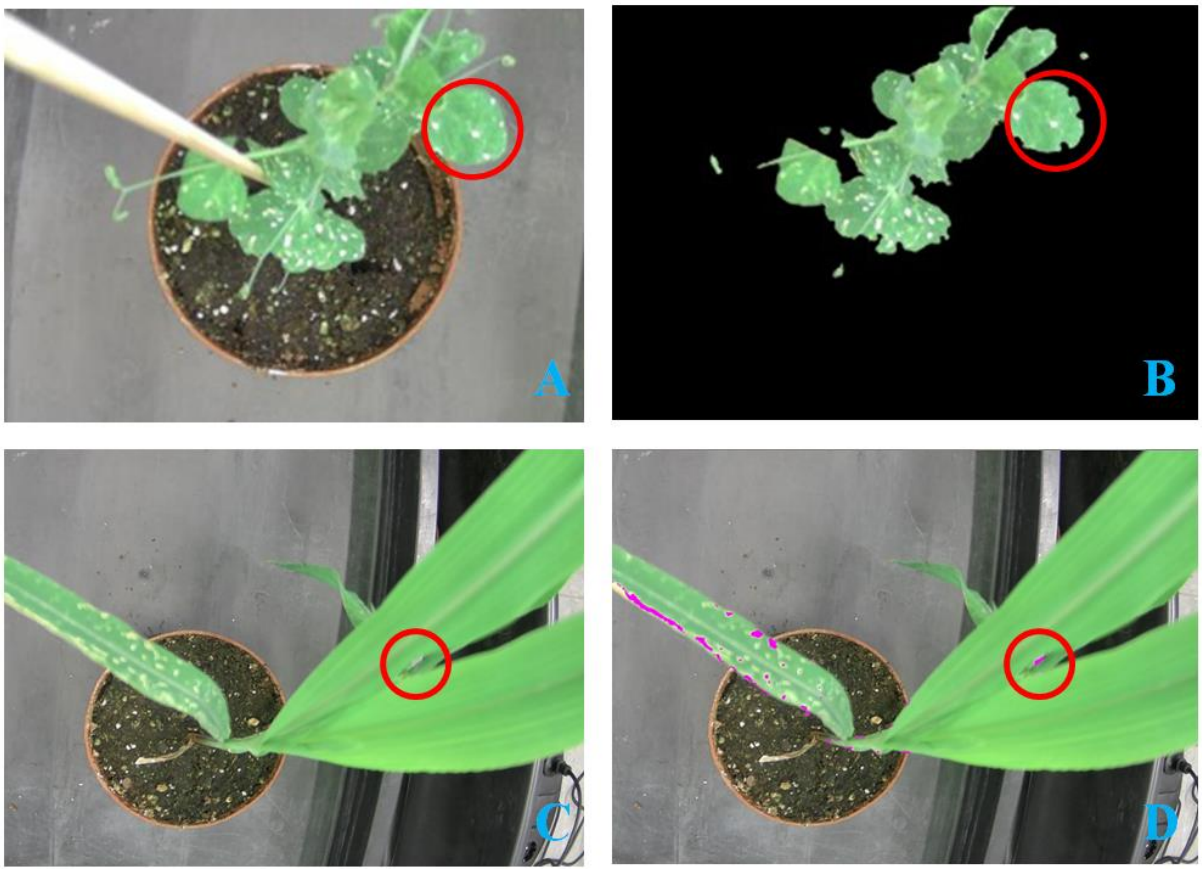


Fig. 14: As visible in image A, leaf-spots at the edge of a leaf did not get included in the plant area image B and thus did not get detected. In image C one can see that on the other hand background areas surrounded by plant-material did falsely get detected as a hit as visible in image D.

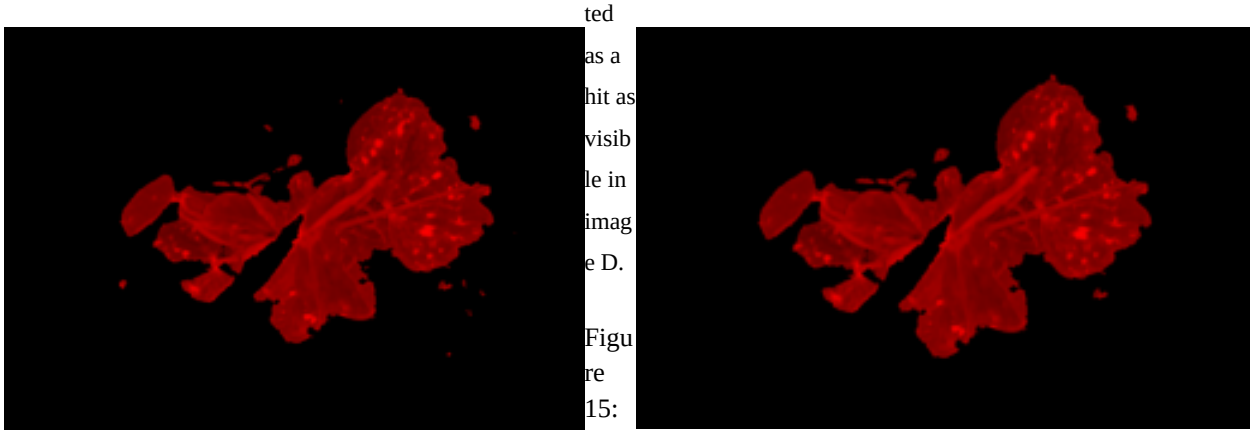


Figure 15: Small spots from greenish perlite and soil surrounding the main structure on the left and on the right the structure after editing the outlines. The remaining spot outside the main structures are the pea's tendrils.

Also, non-plant parts that completely overlap the plant are not excluded due to contouring via outlines and thus later detected as spots, as figure 16 shows.

Figure 16: Parts of the wooden sticks are recognized as spots (shown here in pink) because they are not excluded from mask2 due to the use of outlines.

Spot detection

When looking for spots on the plants, it may happen that plant parts are detected as spots. For example, plant veins may appear white in the light, or plant edges yellowish. Figure 17 shows areas of the leaf margin which are falsely detected in figure 18 parts of the tendrils were detected as spots.

Figure 17: Parts of the leaf margin detected as spots (drawn in pink), Figure 18: Parts of the pea plants tendrils detected as spots (drawn in pink)

To get the best result, we have divided the detection into three different methods. If the intention is to detect simulated pests using red stickers, there is no benefit in detecting other colors than red. On the contrary, here the harm outweighs the benefit, because with all colors also plant parts can be detected. Since red does not occur on the plant, we can offer a very stable procedure.

If the intention is to control the spread of tripses on pea plants, you only need to look for spots in white color area. Noise generated by false positives, because one tries to detect all colors except green, is not useful here either. Since we also detect the colors in HSV Color space, we have the following problem when detecting all colors including white: white in HSV space reflects all color values (H) and is generated only by the low saturation (S) of the colors. If we try to detect a saturation from unsaturated to fully saturated together with all color values, this would produce a lot of noise.

So our three methods are:

1. Detect Red spots: Because the color range is so limited, this works very good without noise
2. Detect everything except green: This finds colored stickers in any color. Because it has a lot yellow color space and a wide saturation range in it, it also detects plants structures sometimes
3. Detect white spots: This will only find small white spots but sometimes also plant structures.

Fortunately, but unplanned, our plants got sick, so we were able to test all three methods.

Figure 19 shows the result of just detecting white spots.

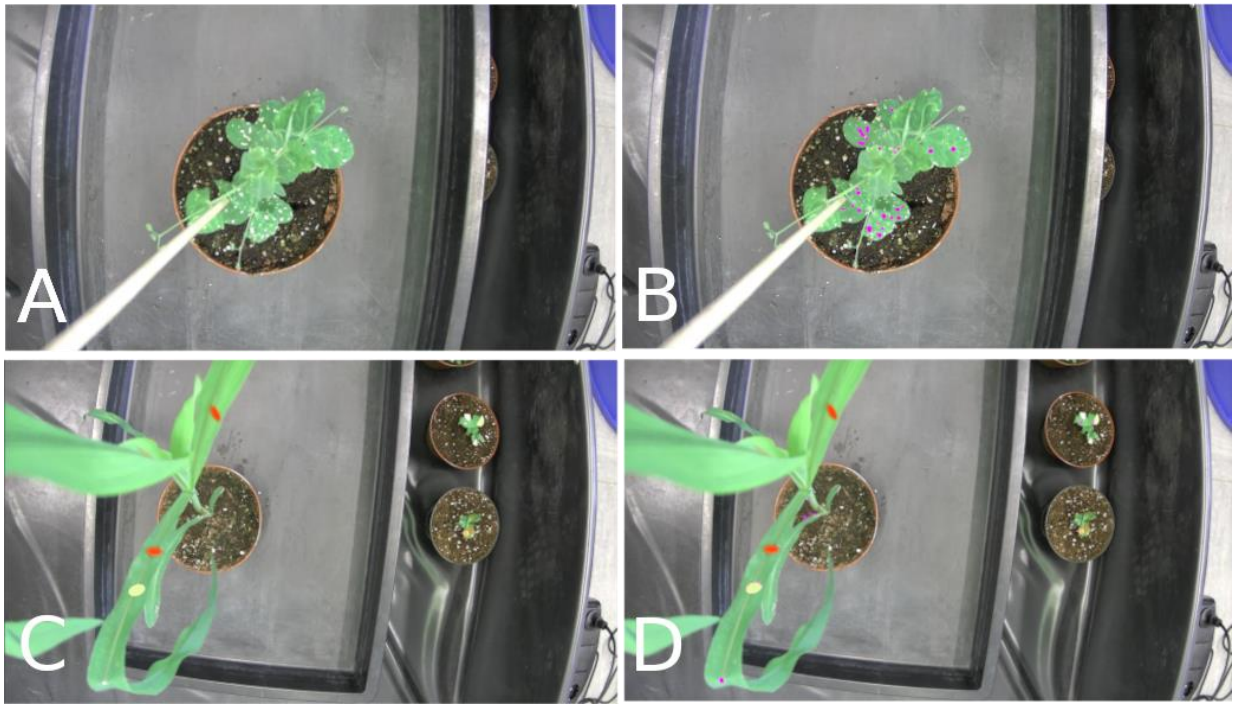


Fig. 19: On the left side (A, C) the images of maize and pea plants taken with the FarmBots Lindy Webcam and processed with the algorithm to detect white leaf spots (pink color) on the right side (B, D).

On Figure 20 the results on just detecting red spot is shown

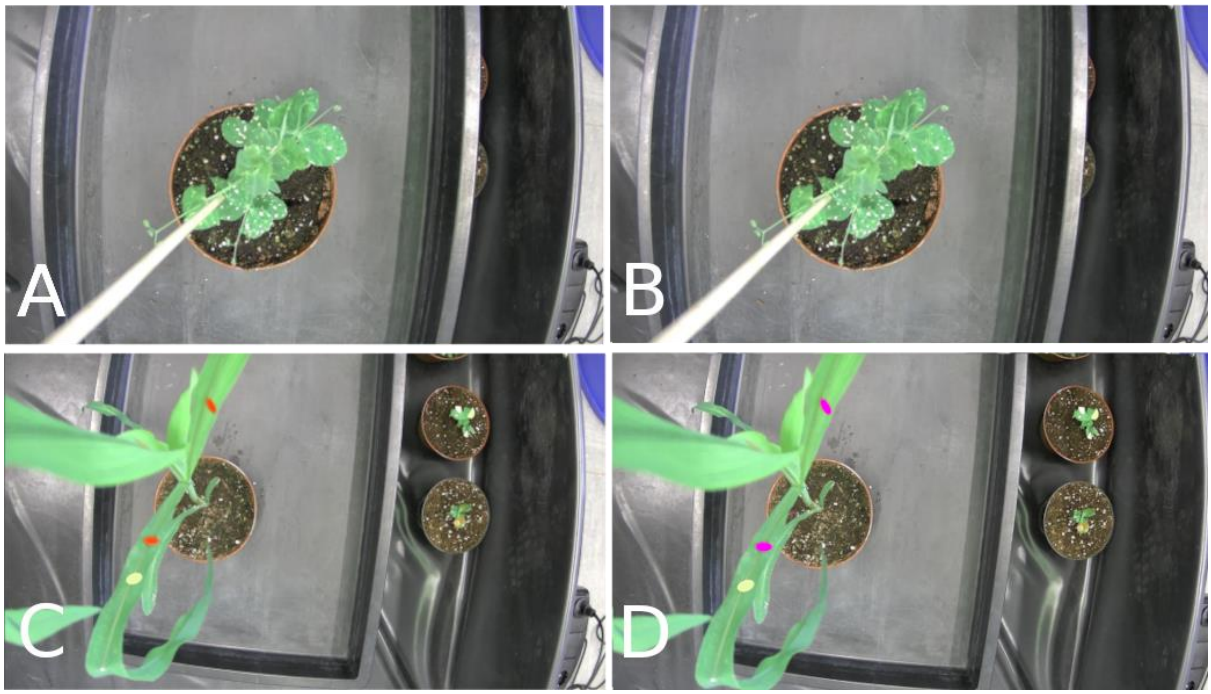


Fig. 20: On the left side (A, C) the images of pea plants and maize taken with the FarmBots Lindy Webcam and processed with the algorithm to detect red leaf spots (pink color) on the right side (B, D).

Figure 21 shows the results on detecting all colors except green with an saturation > 50.

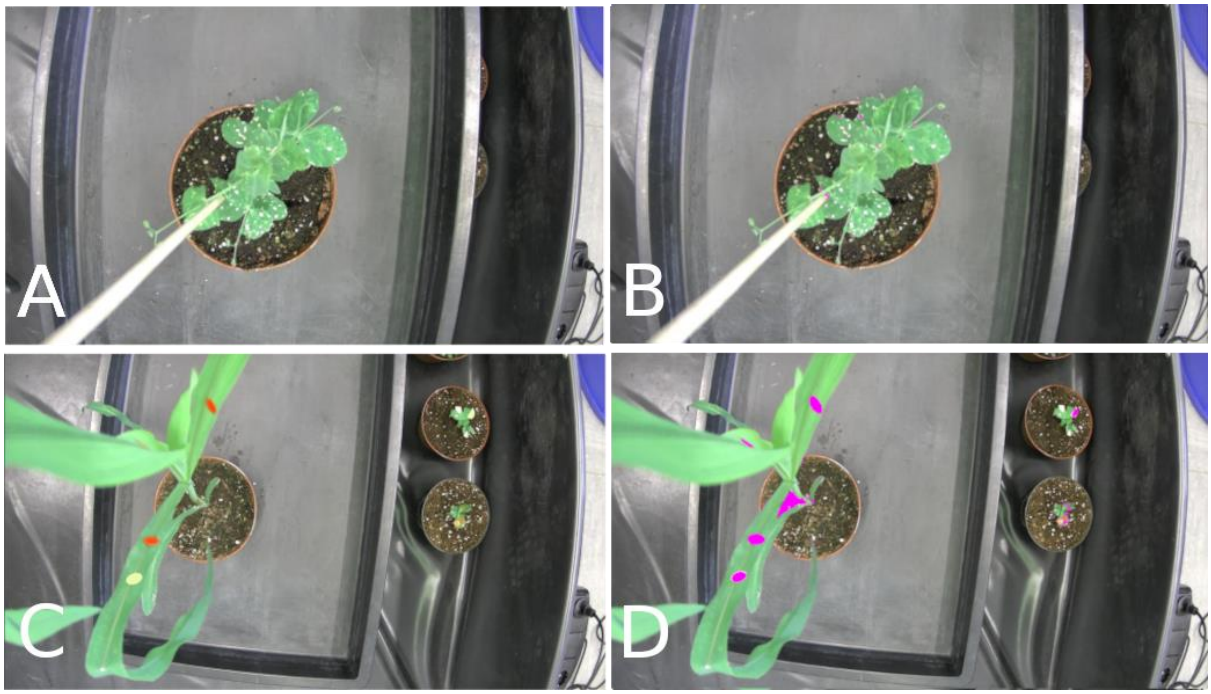


Fig. 21: On the left side (A, C) the images of pea plants and maize taken with the FarmBots Lindy Webcam and processed with the algorithm to detect leaf spots in all colors except green (pink color) on the right side (B, D).

All results can be seen in the appendix.

Discussion

One of the big problems remaining lies in the way the pictures were processed. The approach of cutting out everything outside a green border and searching for anything inside the greed area results in two kinds of faulty detection.

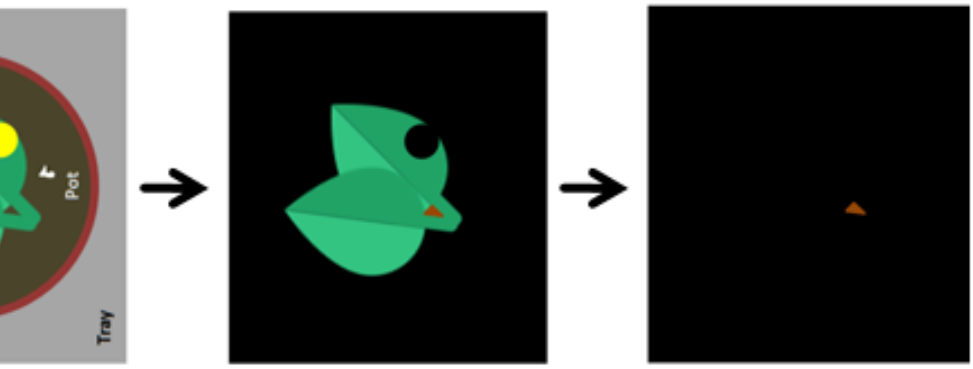


Fig. 22: Schematic of the image processing algorithm (as in Fig. 14) showcasing the two major cases of faulty detection. The first is that leafs-spots at the edge of a leaf result in being cut out as its “outside a green border”. The second is that leafs or plants that are overlapping can create a “green border” around the background that will be detected as a pest.

The first is that leaf-spots on the edge of a leaf will be cut out of the image since it is outside (Fig. 22) the green border and thus wont be detected, a phenomenon one can see in Fig. 14. The second problem originates from plants or leafs overlapping each other, creating a green border around background material that will then be detected later (Fig. 22), a problem also visible in Fig. 14. A possible solution could be not to use the findContours method in outline mode. There are other modes for this method that return all contours plus hierarchy. These found contours then had to be post-processed according to criteria like size, color they

enclose, etc. Implementing this can of course become extremely complex and bring other problems to light. Another possibility would be an application in machine learning.

One important factor on the other hand, significantly affecting the capabilities of the image detection are the lighting conditions. Besides only working at daytime or with adding a lightsource in the field, the warmth of the lightsource in an enclosed room plays an important role. In the camera's perception, warm lighting shifts the green color of the plants towards the yellow spectrum which makes the detection of yellow spots a lot more difficult. For our program to also work in non-ideal light conditions, color correction with a color checker card would be a valuable addition. The idea is to first take a reference picture, with the FarmBot-camera, with the color checker card in frame. The taken picture gets matched to a digital file of the card and the values that were changed are going to get saved, so they can be applied to all the other pictures that are going to get taken in that rotation. This step would be performed before the pest detection part in our code. So after each picture gets taken and matched to the reference picture, it gets analyzed for pests.
Unfortunately we did not have enough time to implement this. But we found a framework called PlantCV, which offers a lot of modular functions for image-processing, that were specifically developed to work on a variety of plants and imaging systems. In this case the Color Correction Workflow fits our needs and can be used as a first reference point for further development.
When choosing a color checker card, there are a lot of different options. For example the 24-Colorchecker passport [Fig 23], which is also used in the PlantCV Color Correction Workflow or a pantone card. For a project like this something more accessible, that can easily be printed out, would be better. We created a 3D-model of a plant marker [Fig. 24], on which the color checker card can be placed on top of and then put into the soil next to the plant. Which makes it easier to take a reference picture from above. Another thing to keep in mind is that this color card approach could take away the automatization characteristic a little bit, since someone would have to place the card into the soil each time before the pictures get taken or at least when the setup changes.

Fig. 23: 24-color Colorchecker Passport

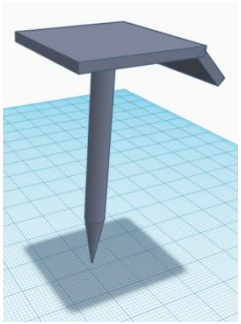


Fig 24: 3D-model of a plant-marker for a color-reference-sheet

Possible integration of our program into the FarmBots general workflow

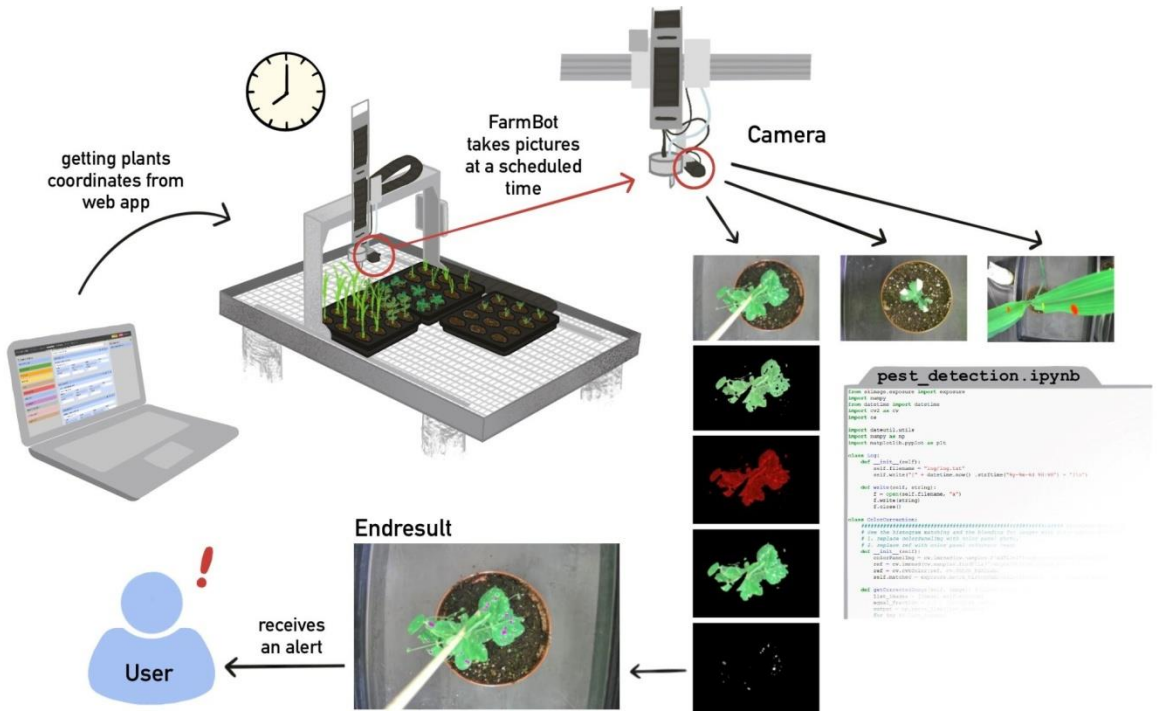


Fig. 25: Visualization of how our program could be implemented into the FarmBots workflow

This Diagramm [Fig. 25] demonstrates a first idea, how our program (pest_detection.ipynb - name of the file) could be implemented into a regular workflow with the FarmBot. First the coordinates of each plant need to be entered and saved on the web app. This has to be done anyway, if the plants are going to get watered by the FarmBot as well. It is also possible for our program to analyze a picture of a whole tray of plants, so less coordinates are necessary. Additionally a time needs to be scheduled. It would be favorable to find out in advance, at what time of the day the lighting conditions are the most ideal to take pictures. There are two different options for taking pictures. The pest detection could be done alongside the watering schedule, so before or after a plant gets watered, a picture gets taken. There might be disadvantages to this option, like time delays. Another option would be a separate schedule for taking pictures. This is more useful if the lighting conditions differ from the watering time or only pictures of the whole tray are needed. The FarmBot obtains the coordinates through the FarmBot API and takes the pictures. The pictures need to be saved at a location, where our program can read from. The code goes through the pictures one by one. The intermediate steps of image processing and pest detection, that are applied to each photo, are shown in the diagram. If a pest gets detected, the user gets notified and receives an alert. It is already implemented that the pixels, which represent the spots, are counted and stored. At the moment these are written to a log file. In the future, these values can be used to trigger some kind of alarm to notify the user. The end result can then be viewed and also saved for later analysis and inspection.

How good was the Farmbot at detecting and should another camera be used, what about the light?

As shown in the results, the webcam of the FarmBot has a high enough resolution even to detect small real leaf-spots that are around 1mm in diameter. Furthermore one can say that the system is already capable of detecting 34 out of 66 actual leaf-spots on the mays-plant (Fig. 14) or 25 out of 61 on the Pea-plant (Fig. 19) and thus would accurately mark the plant as infected for further treatment.

A further interesting development of this project would be to integrate machine learning into the script, which is fed with data from different pests, in order to recognize which pest infected the plants and react accordingly. For a real application for FarmBot users, the entire script should be implemented into the WebApp as an additional option. In this case there could be two interesting options to react to a detected pest. The first would be to add another pump-system to the FarmBot for pesticides. It could then move to the exact location of the pest and treat it locally. The second option would be to mark the infected plants in the map available in the WebApp for a manual interaction.

As already mentioned, localized application of pesticides strongly reduces the amounts used and thus also their negative effects on the soil ecosystem and groundwater. Since commercial agricultural applications exceed the range of any FarmBot, theseable large-scale applications are developed for drones or other robots. On the other hand, the application of an automated pest-detection in the FarmBot could be interesting for research and private users as well as educational programs to raise awareness for the importance of reducing pesticides usage.

References

Borruso, L., Mimmo, T., & al., e. (2021). Next generation biomonitoring to assess key species and soil parameters determining the biodiversity in agricultural soils. *Global Symposium on Soil Biodiversity* (S. 232-234). Italy: Food and Agriculture Organization of the United Nations.

Dean R. Paini, A. W. (5. Jul 2016). Global threat to agriculture from invasive species. *PNAS* , S. 7575-7579.

FarmBot. (2022). *FarmBot at home*. Abgerufen am Feb 2022 von https://farm.bot/pages/farmbot-at-home

Hallberg, G. R. (1989). Pesticide Pollution of Groundwater in the Humid United States. *Agriculture, Ecosystems and Environment* , S. 299-367.

Stoll, A., & Kutzbach, H. D. (2000). Guidance of a Forage Harvester with GPS. *Precision Agriculture* , S. 281-291.

Scikit Histogram. (2022). Last accessed on Mar 2022 by https://scikit-image.org/docs/stable/auto_examples/color_exposure/plot_histogram_matching.html

OpenCV inRange. (2022). Last accessed on Mar 2022 by https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga48af0ab51e36436c5d04340e036ce981

OpenCV bitwise_and. (2022). Last accessed on Mar 2022 by https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga60b4d04b251ba5eb1392c34425497e14

OpenCV getStructuringElement. (2022). Last accessed on Mar 2022 by https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gac342a1bb6eabf6f55c803b09268e36dc

OpenCV split. (2022). Last accessed on Mar 2022 by https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga8027f9deee1e42716be8039e5863fbd9

OpenCV morphologyEx. (2022). Last accessed on Mar 2022 by https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#ga67493776e3ad1a3df63883829375201f

OpenCV findContours. (2022). Last accessed on Mar 2022 by https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a

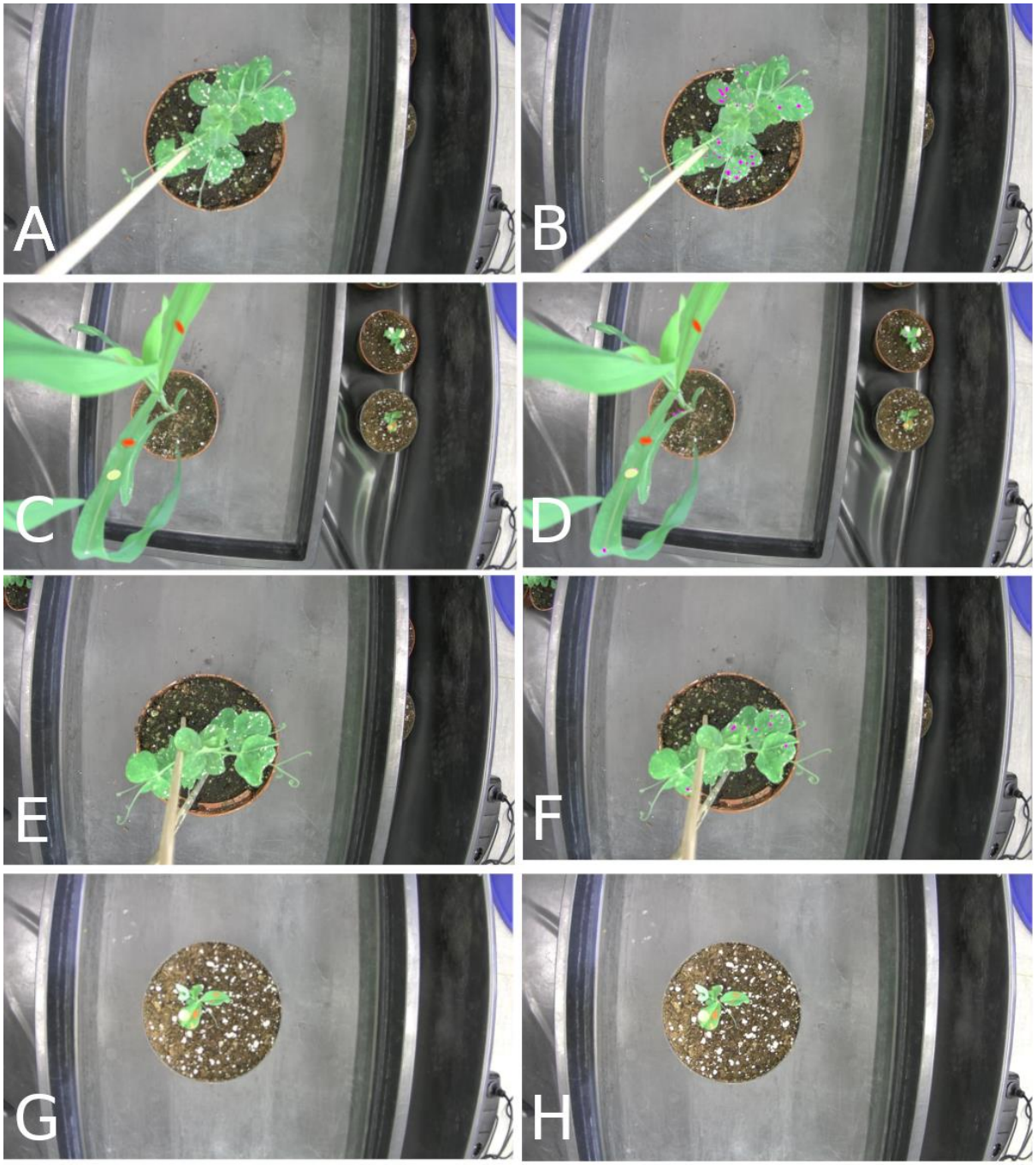
OpenCV threshold. (2022). Last accessed on Mar 2022 by https://docs.opencv.org/3.4/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57

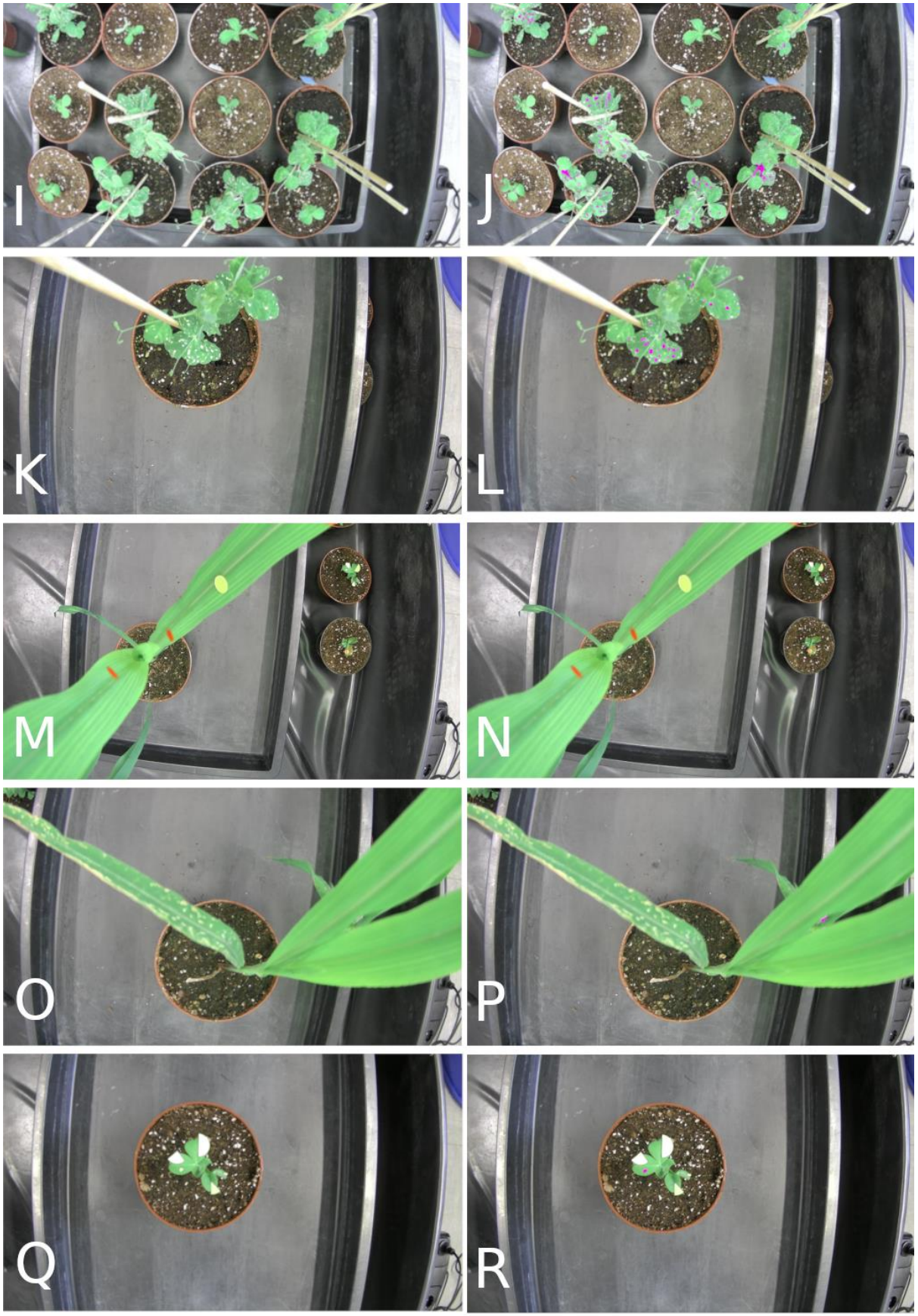
PlantCV Documentation: Color Correction Workflow. (2022). Last accessed on mar 2022 by https://plantcv.readthedocs.io/en/latest/transform_correct_color/

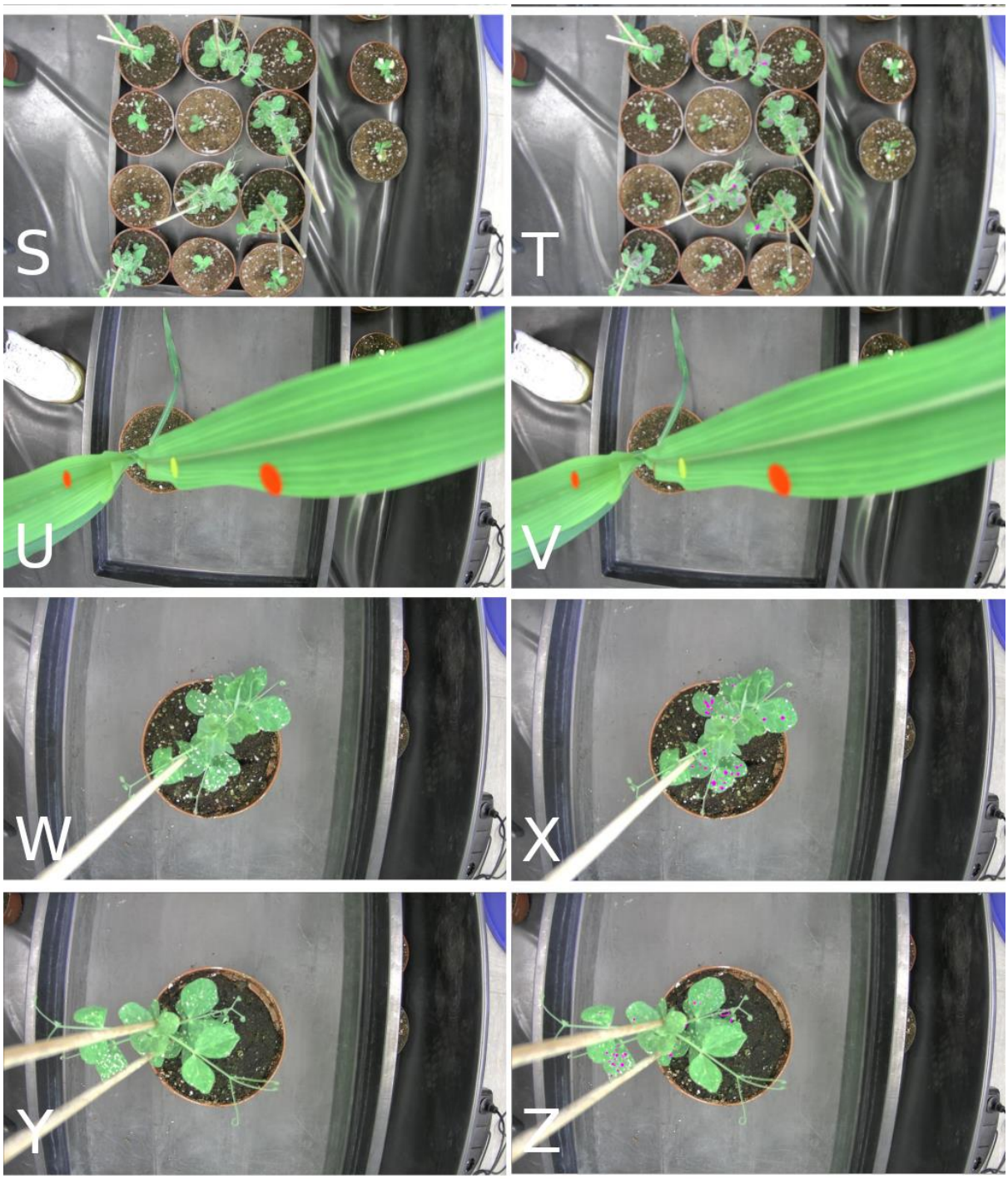
Appendix

1. Full results
    a. Detection of white color
    b. Detection of red color
    c. Detection of every color except green
2. Code

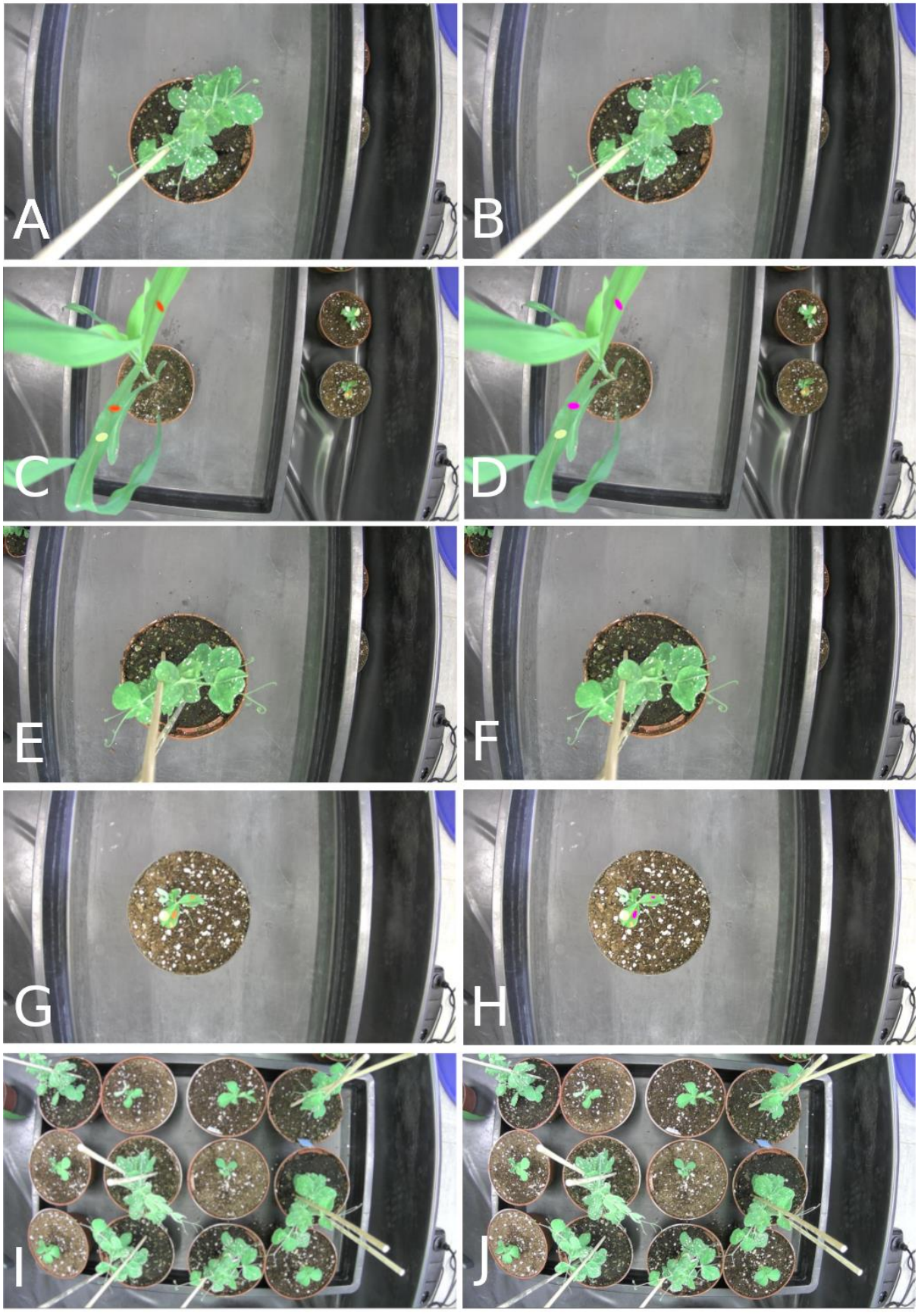
1. Full results
a. Detection of white color

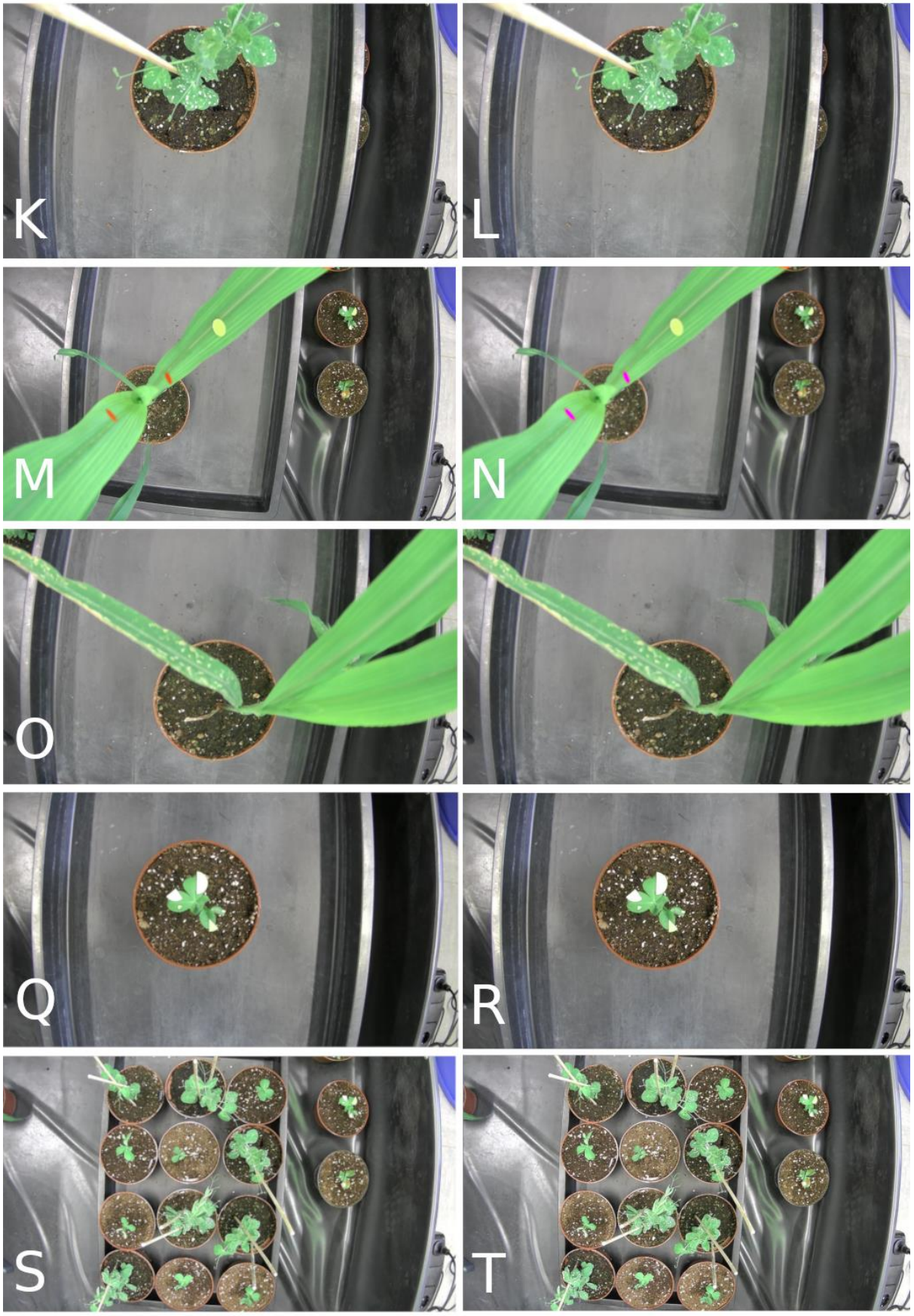

A
B
C
D
E
F
G
H

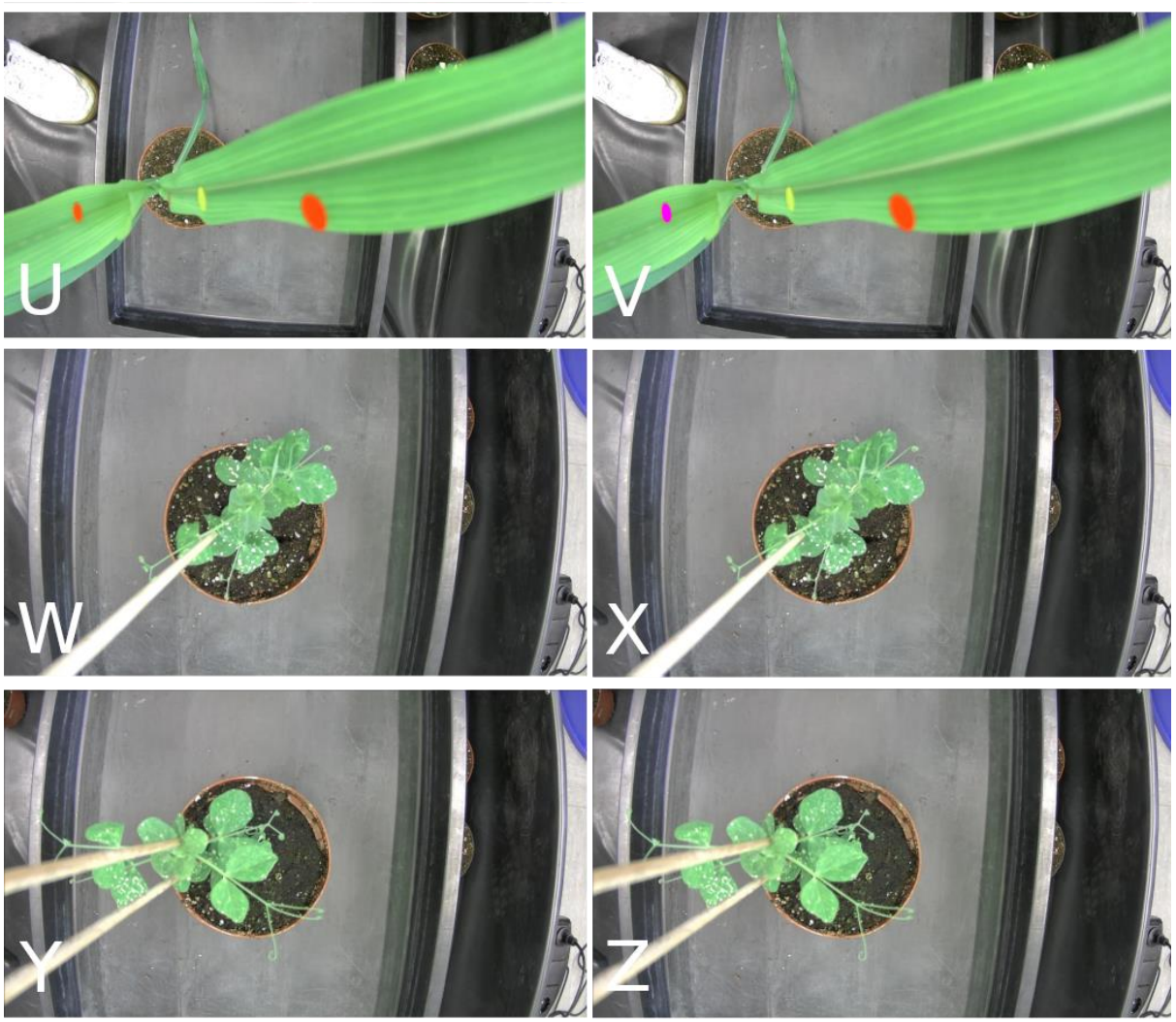

I
J
K
L
M
N
O
P
Q
R



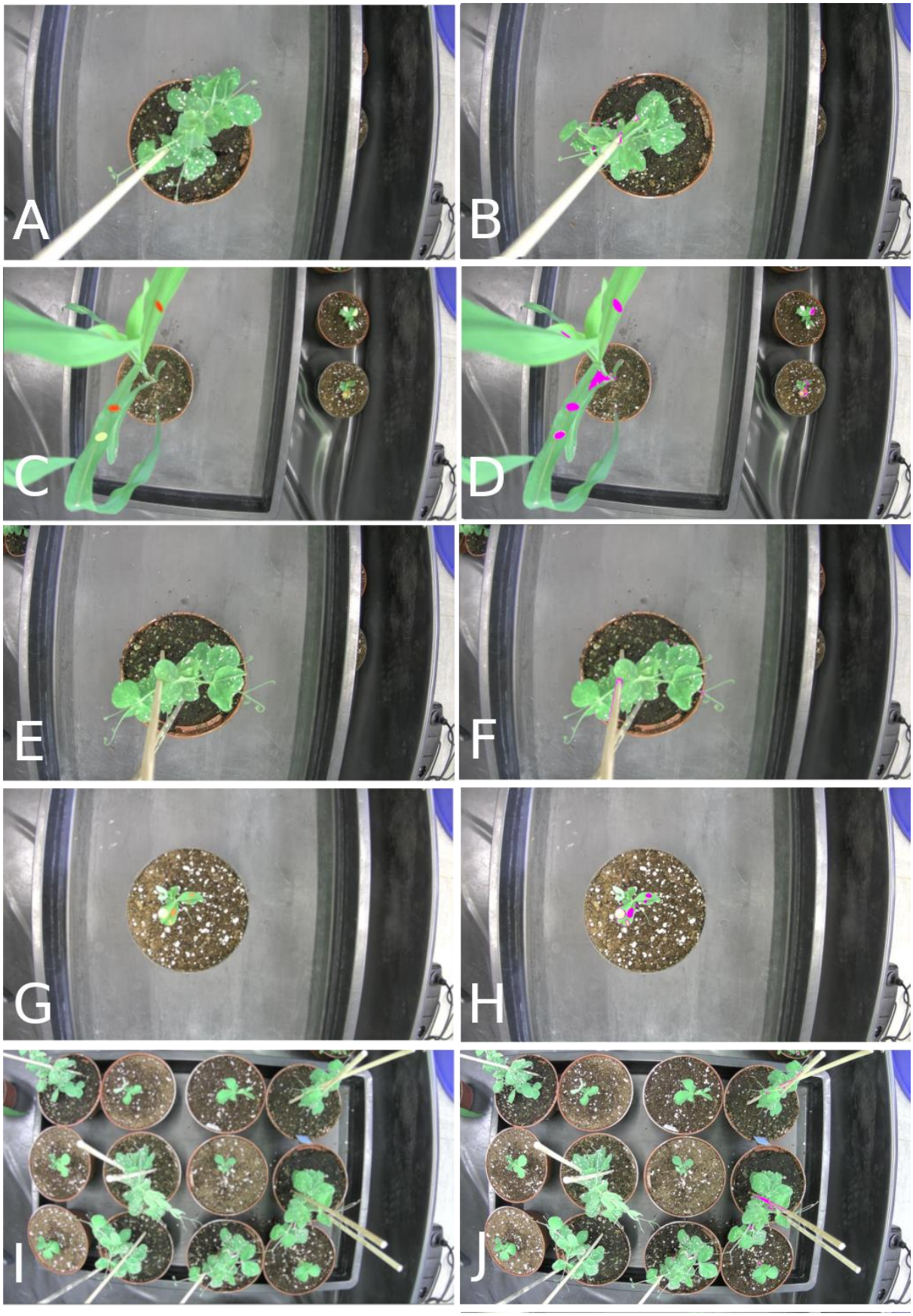
b. Detection of red color

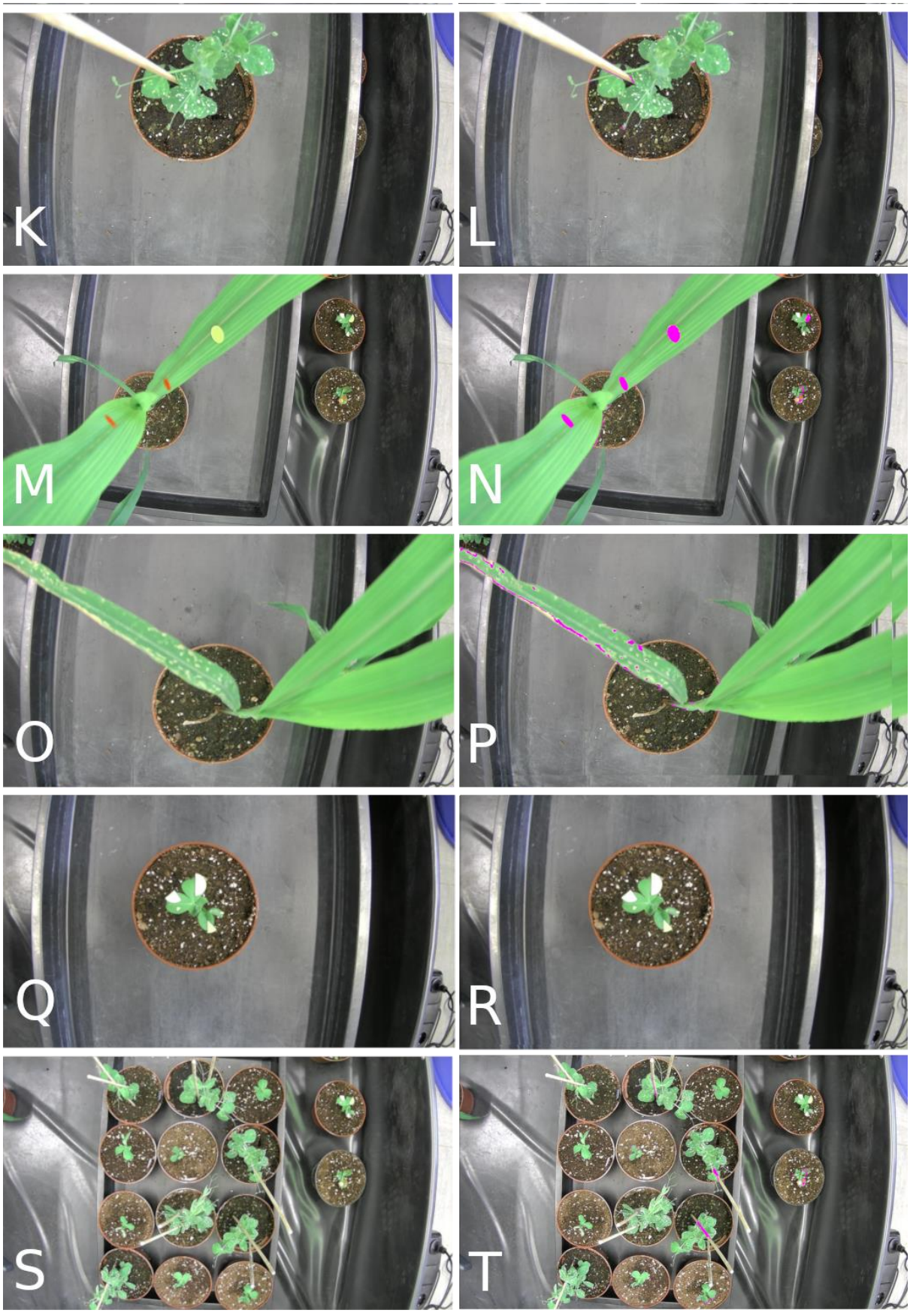

A
B
C
D
E
F
G
H
I
J

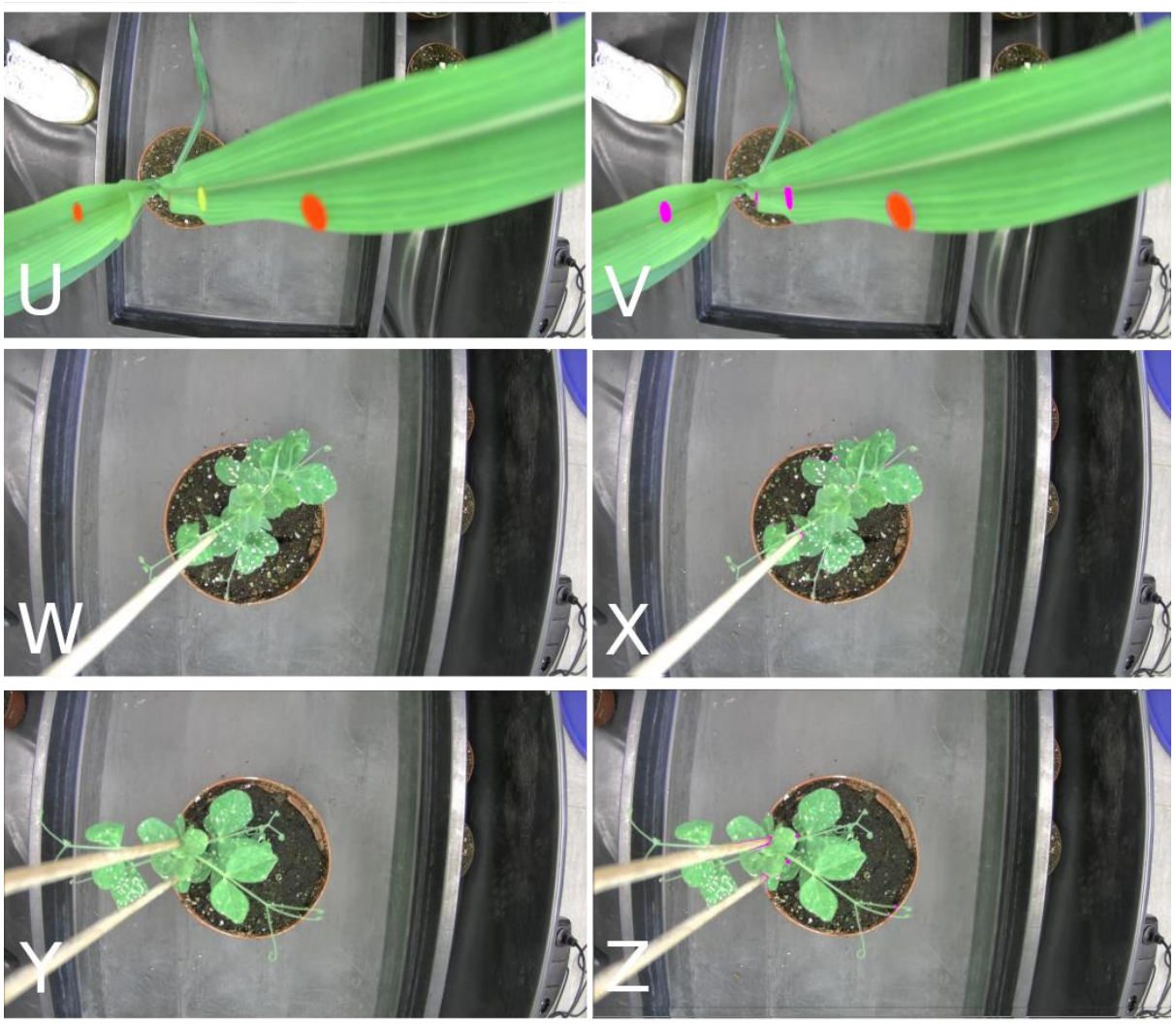

K
L
M
N
O
Q
R
S
T


U
V
W
X
Y
Z

c. Detection of every color except green




K
L
M
N
O
P
Q
R
S
T


U
V
W
X
Y
Z

2. Code

```
#####
# Created By: Ruth Neeßen, Febin Chollapra
# Date: 2022-03-10
# Version = 1.0
# This module detects plants in images, removes the background and finds different colored
# spots on the plants
# #

from skimage.exposure import exposure
from datetime import datetime
import cv2 as cv
import os

import numpy as np
import matplotlib.pyplot as plt

class Log:
    def __init__(self):
        self.filename = "log/log.txt"
        self.write("[" + datetime.now() .strftime("%y-%m-%d %H:%M") + "]\n")

    def write(self, string):
        f = open(self.filename, "a")
        f.write(string)
        f.close()

class ColorCorrection:
   ################## Histogram matching part ######################
    # Use the histogram matching and the blending for images with color panels and work
    # with the blended image
    # 1. replace colorPanelImg with color panel photo,
    # 2. replace ref with color panel reference image
    def __init__(self):
        colorPanelImg = cv.imread(cv.samples.findFile("images/reference/img.png"))
        ref = cv.imread(cv.samples.findFile("images/reference/img.png"))
        ref = cv.cvtColor(ref, cv.COLOR_BGR2RGB)
        self.matched = exposure.match_histograms(colorPanelImg, ref, channel_axis=-1)

    def getCorrectedImage(self, image): # Blend images equally.
        list_images = [image, self.matched]
        equal_fraction = 1.0 / (len(list_images))
        output = np.zeros_like(list_images[0])
        for img in list_images:
            output = output + img * equal_fraction
        output = output.astype(np.uint8)
        return output
```

```
class PlantDetection:

    def __init__(self, folder, filename):
        self.filename = filename;
        self.image = cv.imread(cv.samples.findFile(folder + filename))
        self.image = cv.cvtColor(self.image, cv.COLOR_BGR2RGB)

        ### put color correction back in
        #cc = ColorCorrection();
        #self.image = cv.cvtColor(cc.getCorrectedImage(self.image), cv.COLOR_BGR2RGB)

        #change to HSV color space
        self.image_HSV = cv.cvtColor(self.image,cv.COLOR_RGB2HSV)
        self.plantsImage = None;

    def findPlants(self):
        #green mask, should be adjusted after the implementation of the color panel
        plt.figure(figsize=(20, 42))
        plt.imshow(self.image)
        lower=np.array([36,60,130])
        upper=np.array([86,255,255])
        mask = cv.inRange(self.image_HSV ,lower,upper)
        res = cv.bitwise_and(self.image, self.image,mask=mask)
        blue, green, red = cv.split(res)

        #create closed outlines to create a mask from it
        kernel = cv.getStructuringElement(cv.MORPH_DILATE, (5,5))
        close = cv.morphologyEx(green, cv.MORPH_CLOSE, kernel, iterations=1)
        contours, hierarchy = cv.findContours(close, mode=cv.RETR_EXTERNAL,
method=cv.CHAIN_APPROX_SIMPLE)

        #edit contours and remove smaller spots outside of the plant
        contourList = list(contours)
        contourList2 = list()
        for c in contourList:
            if cv.contourArea(c) > 140.0:
                contourList2.append(c)
        contours2 = tuple(contourList2)
        mask = np.zeros_like(self.image)
        cv.drawContours(image=mask, contours=contours2, contourIdx=-1, color=(255, 0, 0),
thickness=-1, lineType=cv.LINE_AA)

        # output the contoured plants
        #plt.figure(figsize=(10, 22))
        #plt.imshow(cv.bitwise_and(self.image, mask))
```

```
    #create mask from contours via grayscaled image and treshold method and add it
    #to the input picture to get all colors on our plants back in
    img2gray = cv.cvtColor(mask,cv.COLOR_BGR2GRAY)
    ret, mask = cv.threshold(img2gray, 10, 255, cv.THRESH_BINARY)
    self.plantsImage = cv.bitwise_and(self.image,self.image,mask=mask)
    return self.plantsImage;

def findStickers(self, log):

    # get all colors on our plants below green in HSV colorspace
    lower = np.array([0,50,0])
    upper = np.array([45,255,255])
    grid_HSV = cv.cvtColor(self.plantsImage,cv.COLOR_RGB2HSV)

    #mask lower color area
    mask1 = cv.inRange(grid_HSV,lower,upper)

    # get all colors on our plants above green in HSV colorspace
    lower = np.array([86,50,0])
    upper = np.array([179,255,255])

    #mask upper color area
    mask2 = cv.inRange(grid_HSV, lower, upper)

    ##add the two mask to a final mask to get all colors except green on the plants
    mask = cv.bitwise_or(mask1, mask2)
    unique, counts = np.unique(mask, return_counts=True)
    counterAll = 0;
    stickerDict = dict(zip(unique, counts))
    if 255 in stickerDict:
        counterAll = stickerDict[255]
    log.write("Colored found: " + str(counterAll) + "\n")
    res = cv.bitwise_and(self.plantsImage, self.plantsImage,mask=mask)
    return res;

def findRedStickers(self, log):
    # get all colors on our plants below green in HSV colorspace
    lower = np.array([0,100,0])
    upper = np.array([20,255,255])
    grid_HSV = cv.cvtColor(self.plantsImage,cv.COLOR_RGB2HSV)

    #mask lower color area
    mask1 = cv.inRange(grid_HSV,lower,upper)

    # get all colors on our plants above green in HSV colorspace
    lower = np.array([150,100,0])
    upper = np.array([179,255,255])
```

```
        #mask upper color area
        mask2 = cv.inRange(grid_HSV, lower, upper)

        ##add the two mask to a final mask to get all colors except green on the plants
        mask = cv.bitwise_or(mask1, mask2)
        unique, counts = np.unique(mask, return_counts=True)
        counterRed = 0;
        redColorDict = dict(zip(unique, counts))
        if 255 in redColorDict:
            counterRed = redColorDict[255]
        log.write("Red found: " + str(counterRed) + "\n")
        res = cv.bitwise_and(self.plantsImage, self.plantsImage,mask=mask)
        return res;

    def findSmallSpots(self, log):
        grid_HSV = cv.cvtColor(self.plantsImage,cv.COLOR_RGB2HSV)

        mask = cv.inRange(grid_HSV, np.array([0,0,100]), np.array([179,50,255]))
        unique, counts = np.unique(mask, return_counts=True)
        counterSmall = 0;
        smallSpotsDict = dict(zip(unique, counts))
        if 255 in smallSpotsDict:
            counterSmall = smallSpotsDict[255]
        log.write("White found: " + str(counterSmall) + "\n")
        res = cv.bitwise_and(self.plantsImage, self.plantsImage,mask=mask)
        return res;

    # draws the detected spots on the input image
    def showResultImage(self, processedImage):
        img_gray = cv.cvtColor(processedImage, cv.COLOR_RGB2GRAY)
        th, thresh = cv.threshold(img_gray, 0, 255, cv.THRESH_BINARY)
        kernel = cv.getStructuringElement(cv.MORPH_DILATE, (2,2))
        close = cv.morphologyEx(thresh, cv.MORPH_CLOSE, kernel, iterations=1)
        contours, hierarchy = cv.findContours(close, mode=cv.RETR_EXTERNAL,
method=cv.CHAIN_APPROX_SIMPLE)
        cv.drawContours(image=self.image, contours=contours, contourIdx=-1, color=(255, 0,
255), thickness=-1, lineType=cv.LINE_AA)
        plt.figure(figsize=(20, 42))
        plt.imshow(self.image)

folder = "images/input/";
log = Log();

for filename in os.listdir(folder):

    image = PlantDetection(folder, filename)
    image.findPlants()
    log.write("File: " + filename + "\n")
```

```
red = image.findRedStickers(log)
allColors = image.findStickers(log)
white = image.findSmallSpots(log)
image.showResultImage(white)
image.showResultImage(red)
image.showResultImage(allColors)
```

Readme

This module reads all images from the images/input folder. For each image the following steps are processed:

1. The image is buffered in RGB and HSV format.
   `image = PlantDetection(folder, filename)`
2. The outlines of the plants are found via color detection and then the found outlines are edited.
   `image.findPlants()`
3. The search for points on the plant has started. There are three methods for this:
    1. only red points are searched. This procedure is very restrictive.
       `image.findRedStickers(log)`
    2. all colors except green are recognized as points. This procedure is rather inaccurate, since also plant parts can be recognized.
       `image.findStickers(log)`
    3. only white dots are searched for. Here, too, parts of the plant can be falsely detected as dots.
       `image.findSmallSpots(log)`
4. The found points are drawn in pink in the original image
   `image.showResultImage(white)`

For each run, a log file documents which detection method found how many pixels.

Requirements

The following packages are required:

skimage

numpy

datetime

cv2

os

Usage

Load an image

```
image = PlantDetection(folder, filename)
```

Remove everything except the plants from the loaded image

```
image.findPlants()
```

Pepare new logfile entry

```
log.write("File: " + filename + "\n")
```

Find red color in loaded image

```
red = image.findRedStickers(log)
```

Find all colors except green in loaded image

```
allColors = image.findStickers(log)
```

Find white color in loaded image

```
white = image.findSmallSpots(log)
```

Print result

```
image.showResultImage(x)
```

x can be red, allColors or white

Declaration of Contribution

We confirm that our created methods and results are the product of our work and that all additionally used, external resources are declared as such in this manuscript. We also confirm that this is a product of the constructive collaboration of all authors listed above.