THE KOMATSUNA DATASET
WITH
PYTHON AND OPENCV

JULIAN SZACHOWICZ

OCTOBER 30, 2022

Contents

Chapter 1

Introduction

1.1 The Komatsuna dataset

The Komatsuna dataset [1] is an open source collection of exactly 900 images presenting the birds eye view of the growth of six plants with 3 cameras taken over the span of ten days.

Additionally, a set of leaf-labelled images are available. Together these pose an ideal dataset for segmentation with tools such as OpenCV.

Below are examples of two images from the dataset; of a plant and of the labelled leaves.

Figure 1.1: rgb_01_03_009_01

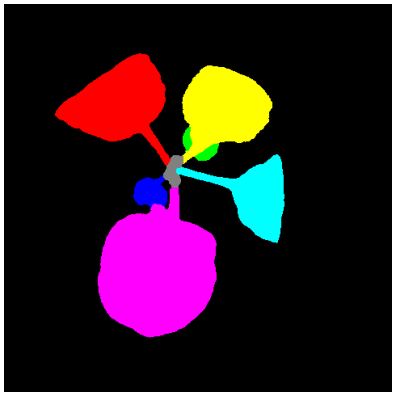

Figure 1.2: label_01_03_009_01

The images have a consistent naming standard which allows easier interpretation of an image. The standard can be found here.

1.2 OpenCV

OpenCV [2] has a vast collection of methodologies which may be utilized, given that a user has in-depth knowledge of their existence.

This project aims to show that a rather simpler approach with a fresh perspective on the possibilities OpenCV may provide can lead to - albeit not perfect, but still satisfactory results to segmentation.

1.3 The Technical Dictionary

This is a list on the algorithms and methods used in the program, together with a short explanation on their nature. It is assumed no theoretical explanation will be required further on besides the following entries.

1. Background subtraction

Also known as foreground detection or segmentation, this method utilizes static cameras to generate a foreground mask by subtraction of images in a set.

Various approaches exist in this area such as the MOG subtractor (Gaussian Mixture-based Background/Foreground Segmentation) or the KNN (K-nearest neighbours Segmentation) - most of which are based on a probabilistic analysis of changing pixel values.

2. HSV filtering

HSV (Hue, Saturation, Value) color space varies from the RGB (Red, Green, Blue) color space in that it approaches color more realistically, neighboring similar, but distorted colors (in saturation or value). In RGB, all color is correlated to the amount of light hitting the object, thus recognition is harder due to the vast difference in two neighboring pixels.

Conversion to HSV allows easier filtering in a range of HSV values and is the go-to method for finding color-specific objects.

HSV filtering in image processing is creating a boolean mask based existence of the pixel in the given bounds.

3. Opening

The basic effect of an opening is somewhat like erosion in that it tends to remove some of the foreground (bright) pixels from the edges of regions of foreground pixels. However it is less destructive than erosion in general. The effect of the operator is to preserve foreground regions that have a similar shape to this structuring element, or that can completely contain the structuring element, while eliminating all other regions of foreground pixels. [3]

4. Connected Components

Computes the connected components in a boolean labeled image. If two neighboring pixels have the same value, they become connected. This is often used to create markers for the borders of each "connected" component.

5. Watershed

Watershed is an approach to object separation based on object shape. The name is based on a geological watershed in which a drainage separates adjacent basins.

Similarly, in image processing watershed algorithm assumes a grayscale image as a topographical surface with minima (basins) and maxima (peaks). Watershed fills in the basins (minima) with "rising water" (labels). Each level of water is another label level.

The traditional approach often leads to over-segmentation. Thankfully OpenCV uses a marker-based approach, which fills known and marked areas of objects with different levels of water.

6. Dice similarity coefficient

The dice similarity index, just as the Jaccard (IoU) index is a statistic used to find the similarity between two samples. In image processing can be used to find the similarity between methods of segmentation by comparing masks.

The formula for the coefficient is as follows:

$$D = \frac{|X \cap Y|}{|X| + |Y|}$$

Chapter 2

Processing

2.1 The Program - a Preface

Due to the complexity of the Komatsuna dataset mainly in terms of lighting variability a more interactive approach to the topic has been developed.

As the entirety of the segmenting process is based on HSV range filtering, the user may use trackbars to observe the influence of change of HSV bounds on the algorithms completing the segmentation.

The program may be found at the following github repository.

The below image presents a view of the program.

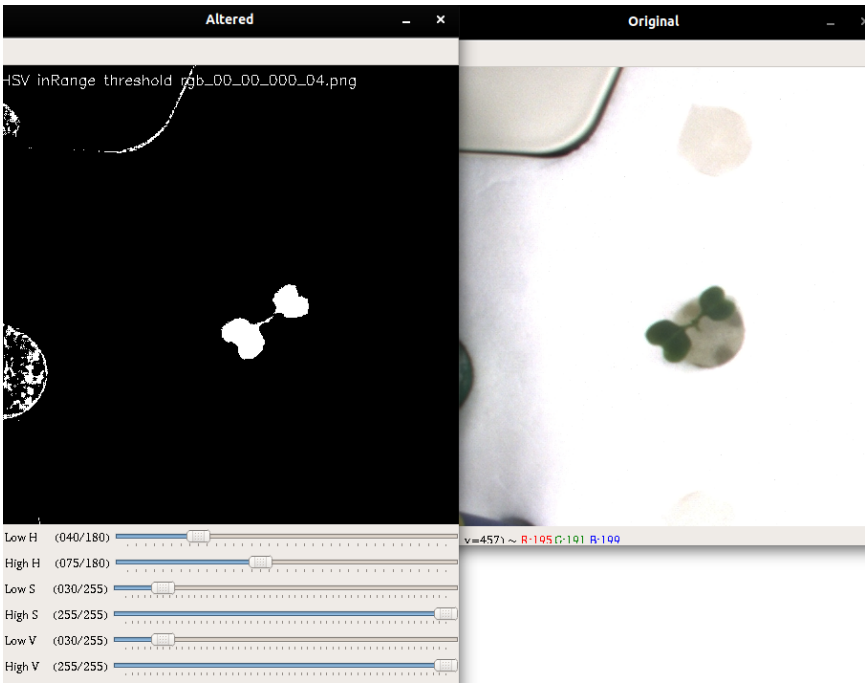

Figure 2.1: Program overview

There are six modes available for analysis and observation.

Mode	Window "Altered"	Window "Original"
0	The default mode; Original	Original
1	HSV filter range mask	Original
2	Watershed mask bare	Leaf Labels
3	Watershed masks with image	Original
4	Sequential background subtraction	Original
5	Sequential backsub with watershed mask	Original

The latter two modes will be discussed later in section 4.3.

The following controls are available for interaction apart from the HSV trackbars:

Key	Function
m	next image
n	prev image
s	save selected image in the selected mode
z	save all images in selected mode
esc	exit the program
d	dynamically calculate dice
f	show dice data based on saved images
1 - 5	select the respective mode

2.2 HSV Filtering

HSV filtering may not be the core of the segmentation processing, but it is as vital as numpy is to OpenCV.

In code, only one line is responsible for this action (see below).

```
1 image_threshold = cv.inRange(image_HSV, (low_H, low_S, low_V),
  ↪  (high_H, high_S, high_V))
```

After trial and error, the default range which somewhat satisfactorily satisfies all 900 images of plants is (40, 30, 30) → (75, 255, 255) (where the maximal values are (180, 255, 255)), but the controls allow setting a more appropriate value.

Below we can observe two images (on the left) for which the default bound values create an appropriate mask.

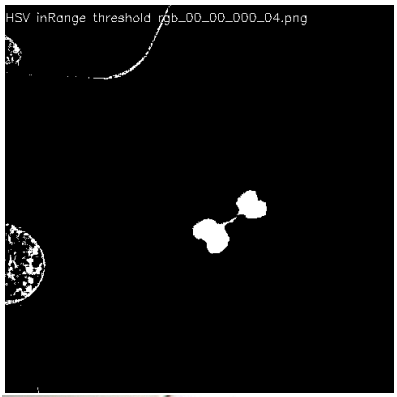

Figure 2.2: hsv_rgb_01_03_009_01

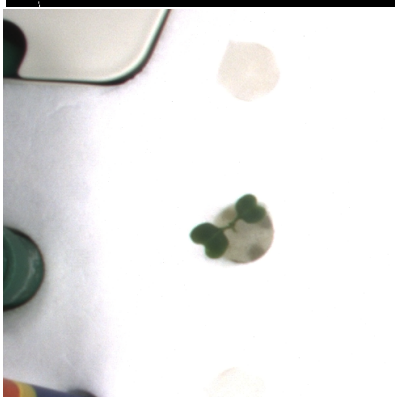

Figure 2.3: rgb_00_00_000_04

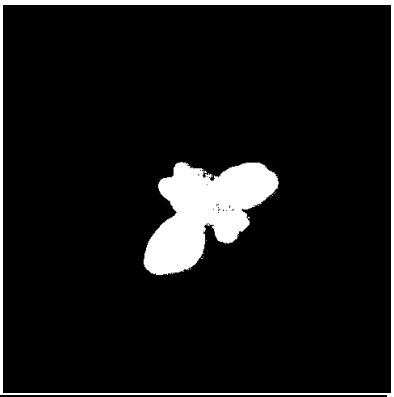

Figure 2.4: hsv_rgb_00_02_004_01

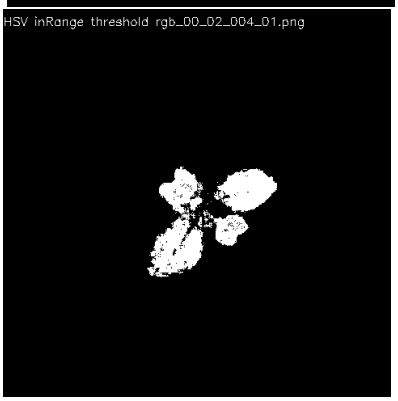

Figure 2.5: rgb_00_02_004_01

But in another case we can observe, that the value is not entirely correct as on the images on the right. Tweaking the slider values may provide a better result. In the last figure we can observe HSV filtering for the range (48, 63, 40) → (180, 44, 155).

And so we unveil the villain of our segmentation project: the plant pot. The shading and color of the pot are too similar to the leaves (especially the lower hanging leaves) and so it is sometimes impossible to eliminate the pot with only HSV range filtering.

There are many methods which could be used to eliminate the pot before HSV filtering such as background subtractions. The theory will be discussed in latter sections. Currently, this simple method, yet flawed method will be used.

2.3 Watersheding

What really is watersheding is a collection of steps finalized with watershed-ing.

First off, the threshold result from HSV filtering is opened and dilated to receive the definite background.

```
kernel = np.ones((3, 3), np.uint8)
opening = cv.morphologyEx(im_threshold, cv.MORPH_OPEN, kernel,
    iterations=2)

sure_bg = cv.dilate(opening, kernel, iterations=7)
```

Based on distance transformation from the opening we may receive the definite foreground by thresholding based on the distance bounds. Additionally, the unknown region is created by subtracting the definite foreground from the definite background.

```
sure_fg = np.uint8(sure_fg)
unknown = cv.subtract(sure_bg, sure_fg)
```

The connected components are then computed to find the markers and filled with the watershed algorithm. Following that, the markers may be used to create mask images [4].

```
_, markers = cv.connectedComponents(sure_fg)
markers = markers + 1
markers[unknown == 255] = 0

markers = cv.watershed(input_im, markers)
```

An example mask as a result may be seen below:

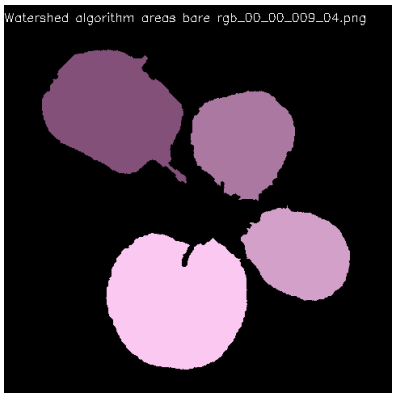

Figure 2.6: mask_00_00_009_04

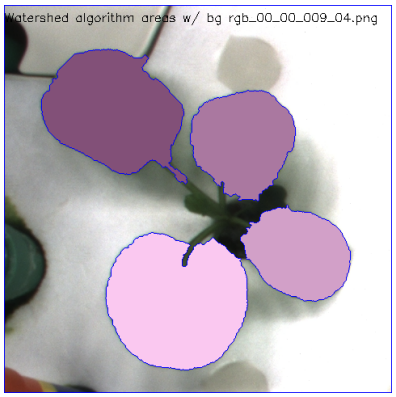

Figure 2.7: rgb_00_00_009_04

Chapter 3

Results

3.1 At the first glance

When taking a first look over the results, one may find many well-fitted masks, such as

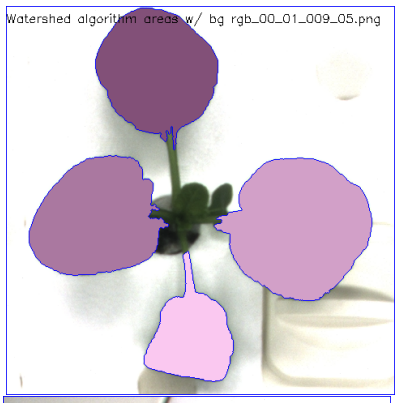

Figure 3.1: rgb_00_00_000_00

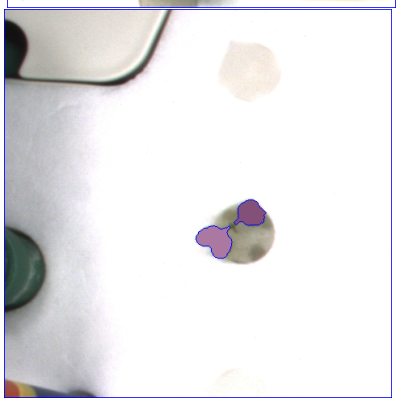

Figure 3.2: rgb_00_01_009_05

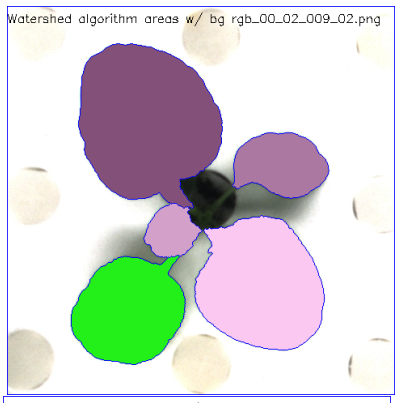

Figure 3.3: rgb_00_02_009_02

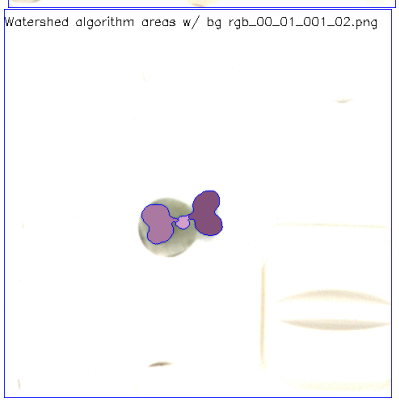

Figure 3.4: rgb_00_01_001_02

But this is quite simply what scientists call optimistic bias. There is a lot of over-segmentation and under-segmentation, which will be analysed in the next sections.

3.2 Over-segmentation

Over-segmentation is the state of detecting parts of leaves as separate leaves. This will most often happen during connecting components, when the leaf veins are improperly detected as borders. This may be due to them being darker or shaded which may cause the leaves to be fragmented and thus cut into smaller pieces.

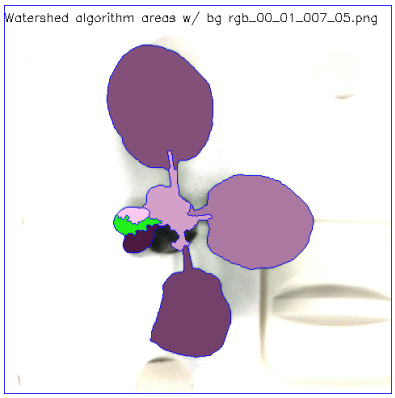

Figure 3.5: rgb_00_01_007_05

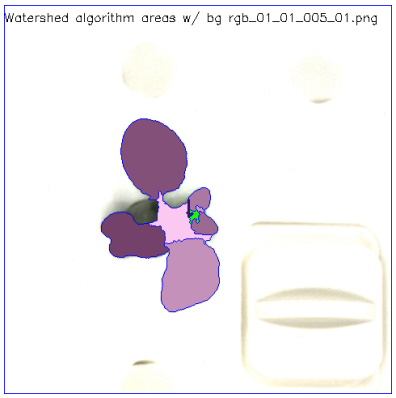

Figure 3.6: rgb_01_01_005_01

3.3 Under-segmentation

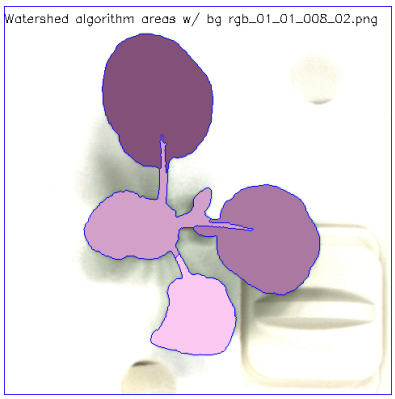

Figure 3.7: rgb_01_01_008_02

Figure 3.8: rgb_01_01_008_02

On the other hand, we may also find cases of under-segmentation, where the algorithm is too lean and lazy. The main culprit here is the aforementioned pot. It falls within the HSV range of the leaves and so is often interpreted as part of the leaf or the leaf itself. Above one can observe the laziness. For the given image rgb_01_01_008_02 the segmentation mask on the left overtakes not only the pot, but also a leaf on the other side.

3.4 Dice Coefficient

Despite these earlier woes, the dice coefficient presents some rather positive results. The overall dice coefficients are presented in the table below.

plant id	mean	min	max
00	0.923	0.685	0.977
01	0.943	0.875	0.978
02	0.927	0.782	0.985
03	0.919	0.718	0.966
04	0.935	0.802	0.969
sm	0.93	0.685	0.985

Where *sm* denotes the summary of all plants and hence images.

As seen, the mean dice result for all 900 images is around 93%. There is no point in going into each image dice value or grouping the means by camera id or day id, other than the one image which should interest us greatly. That is, of course, the image for which we receive the minimal dice coefficient value of 68.5%. This is the mask show below in figure 3.9 with the correct counterpart to the right.

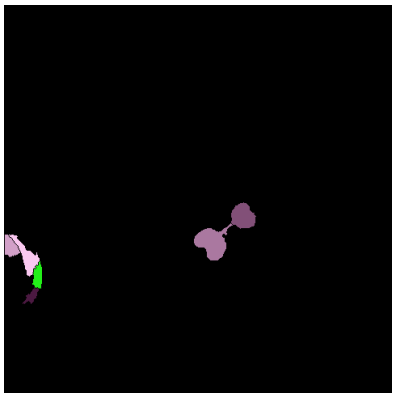

Figure 3.9: rgb_01_00_000_02

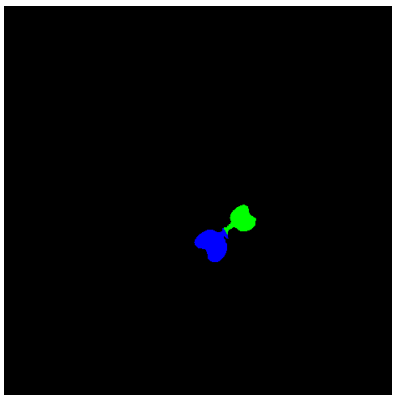

Figure 3.10: label_01_00_000_02

As one may observe the segmentation algorithm was not prepared to entirely remove background which results in certain green-like objects to remain and be segmented, as if they were a leaf.

Chapter 4

Conclusion

4.1 Evaluating the Dice Coefficient

The Coefficient presents valuable information, but as with any statistic, one most be careful with interpretation.

More specifically, in this program, the Dice coefficient is calculated only based on the binary presence of pixels in both labels. It is not observed which mask exists for which leaf, as it has been done with the Komatsuna labels.

In this case, over-segmentation and under-segmentation is still marked as a 1 and thus increases the dice value.

While the coefficient helps in recognizing outliers as have been discussed in section 3.4, it does not in fact mean a segmentation mask is truly as good as manual labelling.

This could be fixed with taking the dice coefficient of each leaf separately, but the algorithm is not adapted to naming leaves, but rather only detecting them. Taking each leaf into consideration would yield the same result, as nameless leaf detection does not take over/under-segmentation into account.

Which brings us to the next section.

4.2 Room for Improvement

Going over the results one can find much space for improvement. The image may be prepped much better with a more altering of the image before or directly after HSV filtering. Unfortunately, many possible approaches would bring their downsides as well. For example:

- Over-segmentation would definitely decrease from the use of various blurring methods such as median blurring or Gauss blurring, but this would lead to more under-segmentation due to fading the borders of components.

- More exact noise reduction could help in removing points falsely recognized as objects, but could lead to dilation of edges of desired objects.
- Enhancing RGB color channels through splitting them and then merging with different strengths may help distinguish the leaves from the pot, but this method is an aggressive approach, often distinguishing parts of leaves from themselves.

Furthermore, the steps before watersheding could be extended by adding steps to recombine connected components which do not fit the leaf profile. Although in most cases this would be a direct road from over-segmentation to under-segmentation.

In any case, this fixes would not address the problem of leaf labelling. The leaves are still treated as detected collections of binary ones and not as distinct objects with different IDs. A different approach to the subject with background subtraction could solve this issue, as explained in the following section.

4.3 Different approach with background subtraction

Background subtraction also known as foreground/background segmentation is a technique used with static cameras as is the case with the Komatsuna dataset. When running the subtractor on a set of camera/plant photos, what we receive is the following (slice of sequential images set).

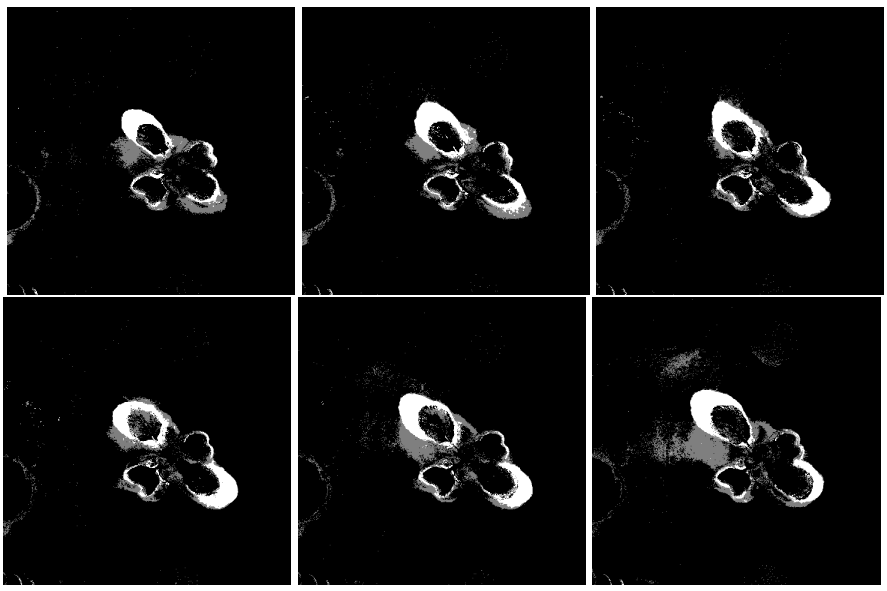

Figure 4.1: (left to right) rgb_00_00_004_02 through rgb_00_00_005_01

And running the algorithm (with HSV range (0, 0, 130) → (180, 255, 255) to filter only white regions and ignore gray "shadow" areas) on this same set results in the following mask sets.

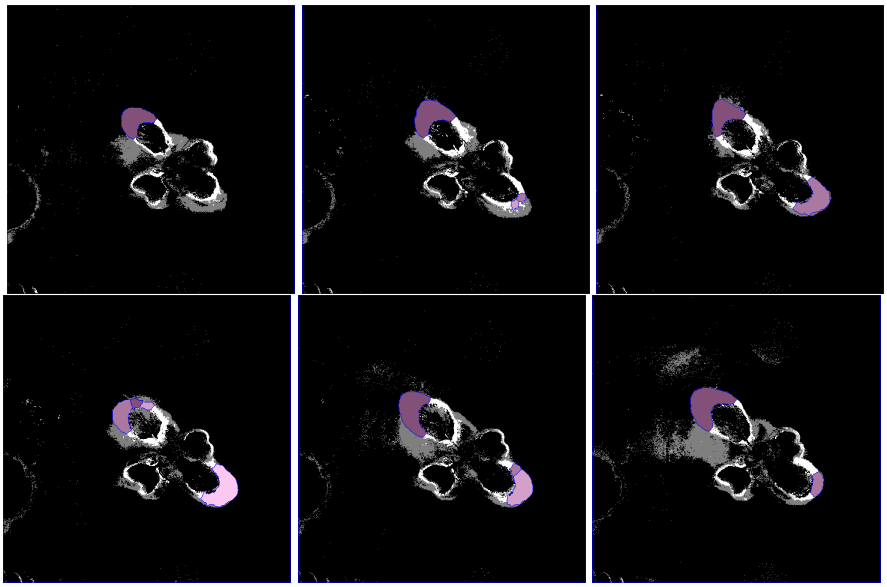

Figure 4.2: (left to right) rgb_00_00_004_02 through rgb_00_00_005_01

With the perfection of the background subtractor, the growth of a leaf may be tracked through the changes brought in each sequential image treating the set of images as a video. The mask could then be theoretically built by combining (or subtracting in certain cases - explained below) each next "growth mask".

Of course this method is not perfect, as the leaves are not the only changing objects in the dataset. Due to variable lighting, static objects are treated as new objects because of their forming in different lighting conditions. An example of objects coming into existence solely due to light change, shown in figures 4.5 and 4.6.

As for the growth itself; it is not always linearly upwards. Sometimes bend under their own weight before they become strong enough to pull upwards. Often other leaves grow at an amzaing rate and cover their comrades leaving them in the shade. An example of a leaf bending, where the masks would have to subtract, shown in figures 4.3 and 4.4.

HSV filtering the image first before going through background subtraction could yield better results.

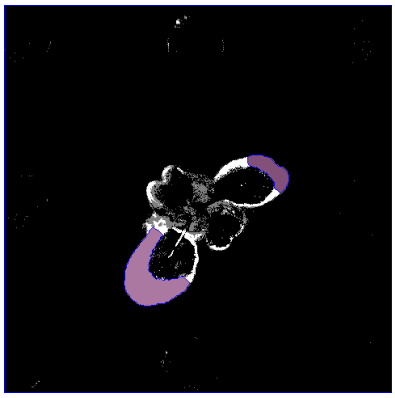

Figure 4.3: rgb_01_02_004_05

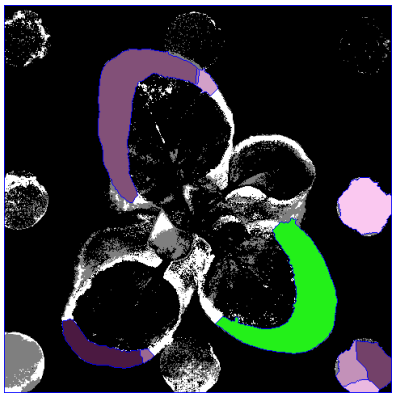

Figure 4.4: rgb_01_02_009_02

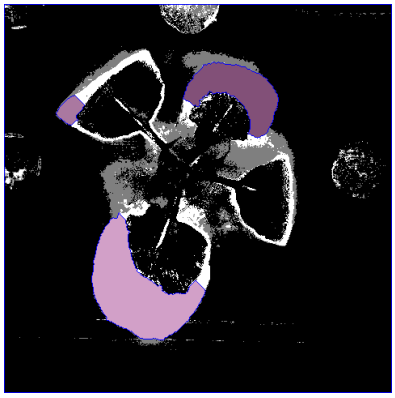

Figure 4.5: rgb_00_03_009_01

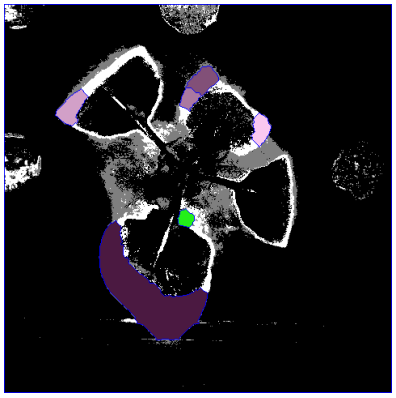

Figure 4.6: rgb_00_03_009_02

4.4 Final thoughts

All in all, a simple approach yields surprisingly positive results. Although the current algorithm (fruit of labour as a result of a one-man hackathon) does not perfectly support all images of the Komatsuna dataset, it lays the foundations for a general approach to segmentation, which may be extended upon in future iterations or similar projects.

Bibliography

[1] KOMATSUNA dataset for instance segmentation, tracking and reconstruction
http://limu.ait.kyushu-u.ac.jp/~agri/komatsuna/

[2] OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library
https://opencv.org/

[3] Opening; Brief Description
https://homepages.inf.ed.ac.uk/rbf/HIPR2/open.htm

[4] Image Segmentation with Watershed Algorithm
https://docs.opencv.org/master/d3/db4/tutorial_py_watershed.html