# Guidelines for image based phenotyping of *A. thaliana*

Leonard Blaschek

June 6, 2021

## Why we use image based phenotyping

We perform image based automated phenotyping to measure more parameters of plant growth at a higher temporal resolution than we could feasibly do using manual methods. Not only do image based approaches allow us to measure plants' height, area, colour and overall shape throughout their life cycle, the high quality images themselves can be used to measure additional features or phenotypes later on. As with all image analysis approaches, the quality of the input images is the single most important variable in determining success. In the approach implemented by me, two imaging setups are used, depending on the developmental stage of the *A. thaliana* plants. In this document, I will try to summarise what you need to know about image acquisition, automated phenotyping and data analysis.

## Contents



# 1 Image acquisition

## 1.1 Rosette growth

From germination until bolting, the plants are best imaged from the top. One big advantage of this is that the pots can be left in the tray, meaning that 15 plants can be analysed from a single picture. As you can see from the example image in figure 1, I acquire the images without labels. Instead, I have labels on the side of the pots (see figure 2). That of course means that you have to be very careful to document the order of pots (genotypes/replicates) in each tray, and orient the trays in the same way in each picture (put a label on one side of the tray to make sure you do not accidentally flip it 180°). You also need to carefully document which picture shows which tray. Lastly, bright markers (any colour that stands out, I use light blue) on the outer edges of the tray allow you to easily crop the image to the area of interest in the analysis. Lastly, taking the pictures always in the same order will save you a lot of time in the analysis (see section 2).

Figure 1: *A. thaliana* plants during their vegetative phase (before bolting of the inflorescence stem), imaged from the top. The image should cover the whole tray and the colour card to the left, both of which should be oriented in parallel with the image borders. Acquisition parameters need to be chosen to avoid over-exposure. Also note the blue stickers on the corners of the tray, which are later used to define the area of interest for the image analysis.

In order to avoid over-exposure, the following acquisition parameters have proven successful in our setup:

- 50 mm objective
- 1/125 s exposure time
- f5.6 aperture
- 800 ISO sensitivity
- white balance set to "Incandescent"

### 1.2 Inflorescence growth

After the plants bolt, we switch to imaging from the front. Now we have to image each plant individually. Make sure that the label on the front of the pot is clearly visible in each image. Everything in the image that is *not* the plant or the colour card, such as labels, sticks, pots, *etc.* should be of a colour that is easy to distinguish from the plant. Because *A. thaliana* can be green (leaves), white (flowers), purple (stressed leaves), yellow/beige/brown (plants in senescence), the ideal colour for foreign objects is blue (see stick in figure 3). Just as with the rosette images, taking the pictures always in the same order will save you a lot of time in the analysis (see section 2).

For this stage, the following acquisition parameters have proven successful in our setup:

- 50 mm objective
- 1/30 s exposure time
- f13 aperture
- 2000 ISO sensitivity
- white balance set to "Incandescent"

## 2 Image metadata

If we want to automate the image analysis, we need to make sure that each image contains all the metadata we need to assign the output to the correct plant. Imagine for example we take an

Figure 2: An *A. thaliana* plant during its reproductive phase (after bolting of the influorescence stem, but **before** binding the stems to a stick), imaged from the front. The image should cover the whole plant and the colour card to the left, both of which should be oriented in parallel with the image borders. Acquisition parameters need to be chosen to avoid over-exposure. Also note the red label on the front of the pot, used for identification. This label can contain any amount of text, but should be blue or red, to avoid colour similarity to the plant itself.

Figure 3: An *A. thaliana* plant during its reproductive phase (**after** binding the stems to a stick), imaged from the front. Just like the label, the stick should ideally be blue to facilitate separation from the plant itself in the analysis.

image of a tray with 15 pots from the top and save that image as `DSC_2487.JPG`. If we input this image into the analysis pipeline, it will output results for each of the 15 plants on the tray, which it will call `DSC_2487.JPG_1, DSC_2487.JPG_2` and so on. But what genotype is plant `DSC_2487.JPG_1` and how old was the plant when the picture was taken? The easiest way to embed metadata in your image is in the image name. If we for example rename our image `DSC_2487.JPG` to `13_2.JPG`, we have the day past germination (13) and the tray number (2) right in the file name.

You can rename the pictures by hand, but if you have followed my advice above and took the pictures always in the same order, we can also automate the renaming. The following script works on MacOS and Linux, and renames all `.JPG` files in the same folder. So create one folder with the picture from the top and one folder with the pictures from the front. The script renames the images with the day past germination and the number of each tray or plant. For example, if your plants germinated roughly on the 24th of July 2020, you'd set `germination=20200824`. If you have three full trays of plants, you would set the `end=3` for the pictures taken from above, and `end=45` for the pictures of the individual plants taken from the front.

```
#!/bin/bash
# define the date of germination for DPG (days past germination) calculation
germination="20200824"
# number of the first sequentially numbered trays/plants
start=1
# number of the last sequentially numbered trays/plants
end=3
# rename images by DPG and tray/plant number
a=$start
for i in *.JPG; do
  var1="_"
  DPG=$(( ($(date -r "$i" +%s) - $(date --date=$germination +%s) )/(60*60*24) ))
  mv -i -- "$i" "$DPG$var1$a.jpg"
  if [ "$a" -lt $end ]
  then
    let a=a+1
  else
    let a=$start
  fi
done
```

On Windows, you could install the bash shell as is described here, to use the above script. Or you use the Python script below

```
## TODO: TRANSLATE BASH SCRIPT TO PYTHON
```

# 3 Image exploration in Fiji

To get a feeling for the images, open one of them in Fiji and explore how you can segment and measure different parts of the plant. A very useful function for this is `Image` → `Adjust` → `Color Threshold...` (use the LAB colour space). To calibrate any height or area measurements, you can define the pixel size using the ruler on the bottom of the colour card.

# 4 High-throughput analysis in Python and PlantCV

You can do the complete analysis in Fiji, but depending on the amount of images you have, this might be prohibitively time consuming. The alternative is a programmatic approach that automates same principles of image analysis that we would use in Fiji. PlantCV is a Python library that was specifically designed for plant phenotyping. Getting started with it is more complicated than using Fiji, but once it is set up the reward is a highly reproducible analysis pipeline that needs minimal manual intervention. In the following I will outline the initial steps of working with our images in PlantCV. Beyond this short tutorial, I highly recommend the excellent official documentation of PlantCV. Lastly, if you are already familiar with PlantCV and want to skip ahead to the analysis pipeline optimised for our imaging setup, head over to my Github page.

## 4.1 Installation

I will assume that most students reading this document are using Windows 10. But even if not, most of what follows is operating system agnostic and will work (almost) identically on MacOS and Linux.

**Install Anaconda** `Anaconda` and its slimmed down version `Miniconda` are package and environment managers for Python. It allows you to create virtual environments for specific tasks, which simplifies work in Python immensely. If you do not have it installed already, I recommend `Miniconda`. Simply follow the installation instruction for you operating system. If you are unsure, your operating system is almost definitely the 64-bit variant.

**Install PlantCV** If you are on Windows, start the `Anaconda Prompt`, on MacOS and Linux simply open a terminal window. To test that conda is properly installed, type:

```
conda --version
```

The prompt should then output the installed conda version, for example `conda 4.9.2`. If this worked as expected, we next need to add the channel containing the PlantCV package to the conda configuration:

```
conda config --add channels conda-forge
```

Now we install PlantCV in a new environment called `plantcv`:

```
conda create -n plantcv plantcv
```

Last we will activate our new environment and install some additional packages:

```
conda activate plantcv # activate the plantcv conda environment
conda install -c conda-forge jupyterlab # install JupyterLab
conda install -c conda-forge imutils # install imutils
```

## 4.2 Optimisation in JupyterLab

To see how well my (or any other) image analysis pipeline works for your specific images, first try it out on single images in `JupyterLab`. The advantage of this approach is that you get intermediate outputs at every step of the pipeline, easily letting you pinpoint any sections that need adjustment. First, make sure you are in the correct conda environment, then start up `JupyterLab`:

```
conda activate plantcv # make sure you are in the correct conda environment
jupyter-lab # start JupyterLab
```

`JupyterLab` will start in your default browser, check for a newly opened tab if you already had a browser window open.

### 4.2.1 Rosette growth

Lets start with the images from the top. Download the `JupyterLab` file tray_vis_allow_gaps.ipynb, save it somewhere convenient (I would advise creating a dedicated `PlantCV` folder), and open it in JupyterLab. As you see, the code is split up in individual cells that can be run one at a time.

**Cell 1** The first cell loads all necessary packages and defines that the intermediate output is directly printed into the JupterLab notebook.

**Cell 2** In the second chunk, you need to change the path in the `pcv.readimage` function so that it leads to a representative picture of your plants. I would choose a full tray at an intermediate growth stage. *There are some parts that are commented out which relate to colour correction. Ignore those for now.*

**Cells 3–5** The next three chunks identify the colour card, write it's size to file as a scale reference, and crop out the colour card so it doesn't interfere with the downstream analysis. *Again, ignore the commented-out lines about colour correction.*

**Cell 6** Next, we identify the bright markers in the corners of the tray and crop the picture accordingly.

**Cell 7** This cell simply outputs the L, A and B channels of the LAB colourspace, and the H and S channels of the HSV colourspace (the V channel is roughly identical to the L channel of the LAB colourspace). The output purely serves to give you a better idea of which of these channels are suitable for segmentation. What makes a channel suitable for segmentation is the difference between the plant and the background. In most scenarios, the A and B channels will work best (figure 4), but depending on your specific case, the others might be useful as well.

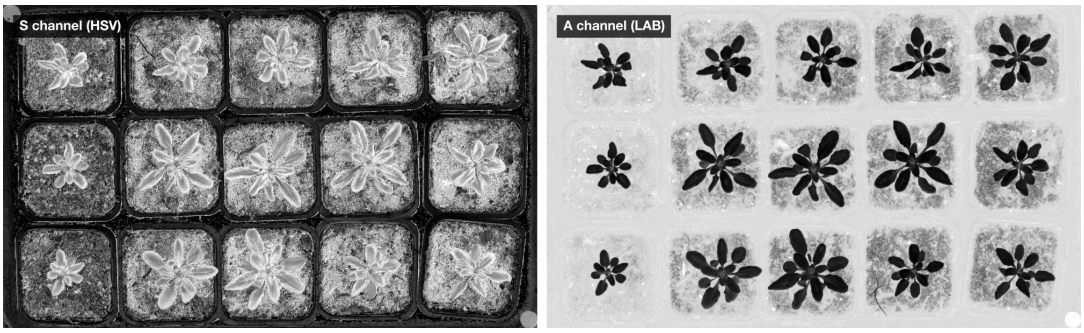


Figure 4: The saturation (S) and red–green (A) channels from the HSV and LAB colourspaces respectively. The S channel is clearly not very useful in this example, while the clear difference between plant and background in the A channel makes the latter a prime candidate for image segmentation.

**Cell 8** This is the most important cell of the pipeline. Here, you use the channels identified in the previous cell to reliably segment the image so that only the plant remains, and the background is removed. In the file you downloaded, I use channels A, B, and H. For each of these, I create a threshold defining which pixels are classified as plant, and which as background. We then join these thresholds, only keeping the pixels that were classified as plant in all three channels.

After the thresholds have been applied, we filter and clean the image a little to remove any small incorrectly classified outliers. the `if--else` statement here relies on the consistent naming of your files, because it changes parameters depending on plant age. In early stages, we remove only very tiny outliers, because the plant itself is very small. In later growth stages, we can remove larger outliers without risking losing the plant.

**Cells 9–13** Here we cluster the pixels classified as plant into individual plants, and export one image for each plant. This is just so you have the segmented images for presentation purposes. If you took good pictures and chose the right thresholds in cell 8, the output of cell 12 should look like figure 5.

Figure 5: Successful clustering of plant-pixels into individual plants. Note some minor miss-classifications like the string shaped contour in the red plant, which will be filtered out in the coming steps.

**Cell 14** This is the actual image analysis part. The function takes our segmented image, defines the regions-of-interest (ROIs) where it expects plants, and analyses each plant. It outputs parameters of rosette shape (area, compactness), colour (HSV), and morphology (leaf count) to `.json` files. The analysis also allows for holes, *i.e.* trays with less than 15 pots or pots with no plants, without messing up your plant numbering.

**Final touches** Now the pipeline is optimised for the image you chose in the beginning. To make sure that it will work for your whole dataset, go through the JupyterLab notebook with other images, especially those from very early and very late stages. Adjust the thresholds in cell 8 to get the best results across your dataset. In early stages, too stringent thresholds will lead to the loss of leaves or whole plants, while the biggest problem in later stages is plant overlap. If two plants overlap, the pipeline will classify them as one plant, messing up your results. The only remedy for that is to open the respective images in any image manipulation software (Fiji/GIMP/Paint), and separating the plants by drawing black lines across their point of overlap.

### 4.2.2 Inflorescence growth

The pipeline for the inflorescence growth, *i.e.* the pictures taken from the side, is very similar to the one for the rosette growth. Download the plantcv_vis.ipynb file, and open it as described above. Below, I will highlight some parts that are different from the rosette pipeline or might need tweaking.

**Cell 3** At some point in later growth stages, you probably started tilting the camera by 90° to get the whole plant on the picture. In this cell, we detect whether the image was tilted to the left by checking the position of the colour card. If the image is tilted, we rotate it to the correct orientation and define a variable called `rotated` for use in coming cells. *Note: if you tilted the camera to the right, we will need to think of another way to correct for that. Let me know.*

**Cell 6** This cell is equivalent to cell 8 in the rosette growth pipeline. The main difference is twofold. Firstly I also leverage the L channel here, because our background is uniformly black. Secondly, I define the hue threshold from two sides, a low and a high part. Combining these two thresholds with the `pcv.logical_or()` function filters out only the blue, which is in the middle of the hue scale. Adjust the thresholds to fit your pictures.

**Cell 8** Here we define a static ROI. There are two sets of coordinates, depending on whether we rotated the picture previously or not. Only pixel clusters that are at least partly within this ROI will be used for analysis. The main use for this step is to filter out leaves, silliques and other debris that is on the ground. This means that the most important part of the ROI is its lower edge. The other thing to consider is that if you choose the ROI too small, imperfect segmentation can lead to parts of the plant being discarded when their connection to the rest of the plant is interrupted.

**Cell 11** This is the image analysis part. It outputs essentially the same data as you've seen for the rosette growth pipeline. If everything worked as it should, the output of this chunk should look like figure 6.

**Final touches** As with the pipeline for the rosette growth, it is important to test your adjustments with multiple images. Especially test images of older plants, because senescence changes the overall colour of the plants quite dramatically.

## 4.3 Batch analysis of multiple files

## 4.4 Data analysis

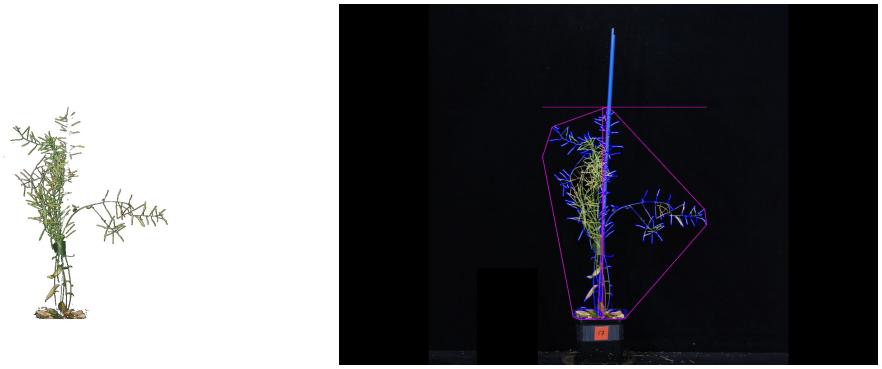

Figure 6: Successful segmentation of an inflorescence stem. In the right image, the convex hull of the plant is delineated in purple and the plant outline in blue.