Plant Senescence Phenotyping via Automated Colourimetric Assay (ACA)

Stefan Bieker
Justine Bresson

ver. 0.04, June 21, 2017

Introduction

When phenotyping leaf senescence, leaf colouration is a commonly used parameter to quickly estimate senescence induction and progression. Usually leaves are ranked based on their degree of yellowing. Starting with green colouration, the leaves will turn yellow and eventually brown. In that way, leaf ageing can be monitored via leaf colour. However, colour perception is always very subjective and context dependent. Furthermore, it is difficult to assign a general colour to leaves presenting colour heterogeneity during senescence. All this will lead to substantial bias when scoring leaves based on their colouration. For this reason we developed an automated and reproducible approach to identify and quantify colours in pictures of single leaves in the context of senescence progression analysis called *Automated Colourimetric Assay* (ACA). This approach is based on two separate scripts written for *ImageJ* and *R* (Schindelin *et al.*, 2015 and R Core Team, 2016).
The first part is an *ImageJ* macro designed to semi-automatically cut and export images of single leaves from an image of a dissected rosette. The second part is an *R* script to categorize the cut leaves. Here, leaves are analysed for the fraction of green, green-yellow, yellow, brown and purple pixels.
Each part will be described in the following with examples and advice for the usage. Examples are derived from the data published in the main publication.

ImageJ Script

Although we are currently working on an automated color segmentation of the pictures, for the analysis with ACA, at the moment it is still necessary to have pictures of single leaves to feed them into the script. Manually cutting and exporting images can be a tedious and lengthy procedure. To allow efficient cutting and background cleaning of the acquired images, we devised an *ImageJ* script (here used v. 1.51j). This script is designed to automatically name and export selections of leaves. The chosen file format is PNG, although other formats are supported by *ImageJ* and may be chosen as well, we strongly disencourage using JPEG-format. When using JPEG, especially in the cleared regions colour distortions are observed leading to very high values of *Unknown*-pixels in the later analysis.
Figure 1 shows an exemplaric image of a dissected rosette during a phenotyping experiment. A black background was chosen to avoid colour-distortions due to for example light reflecting from a white background. Also, the leaves were fixed with double-sided sticky tape to the background to have a flat image and prevent curling. This is not mandatory, still it allows to exactly measure the leaf area later on. Furthermore, senescence is induced at the leaf-margins, which would not be visible when taking a picture of downward curled leaves. Image acquisition must be carried out under standardised conditions. Although colours in general will be recognized correctly even with variations in lighting, distinguishing between e.g. closely related colourtones will not work if lighting varies during image acquisition. This is why we recommend to either use a photochamber and zenithal imaging, where lighting is constant or to use a scanner.

Procedure

- Copy & Paste the script into a .txt file.
- Install the script via the *Plugins\Macros\Install...* tab.
- Start the macro.
- The macro will ask for:
    - two folders. One containing the input images, one to save the output files to.
    - the number of rosettes in the currently chosen image.
    - the plant identifier (e.g. genotype, number etc.).
    - the starting leaf number.
    - the total number of leaves to extract from the currently chosen rosette.
- Select first leaf with the magic wand. Press OK.

For the in figure 1 displayed rosette, the plant name would be *65*, the starting leaf number *3* (notice the two crosses for the missing leaves) and the number of leaves to extract would be *8*. The *starting leave number* may also be used to continue an analysis, which was interrupted before. The saved files will be named *'Plant_65_Leaf_X.png'*.

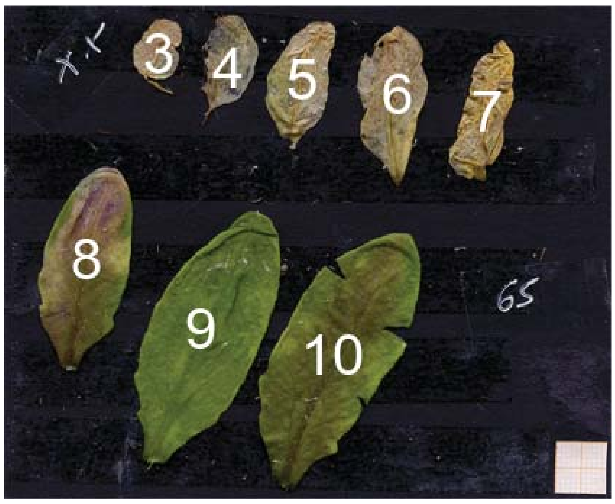


Figure 1: Example scan of a dissected rosette. White numbers indicate the corresponding leaf number. Two crosses mark missing leaves.

```
  ///////////////////////
  // ImageJ ACA script //
  ///////////////////////

// GET IMAGES AND OUTPUT FOLDERS
Dialog.create("Macro extract individual rosette data");
Dialog.addMessage("Create two folders:");
Dialog.addMessage("\n");
Dialog.addMessage("image input folder");
Dialog.addMessage("\n");
Dialog.addMessage("image output folder");
Dialog.show();
inImage = getDirectory("Choose the image input folder");
outImage = getDirectory("Choose the image output folder");
images = getFileList(inImage);

// START NEW OR CONTINUE SELECTION
pathContinued = outImage + "Log.txt";
if(File.exists(pathContinued)) {
    filestring=File.openAsString(pathContinued);
    rows=split(filestring, "\n");
    x=newArray(rows.length);
    for(b=0; b<rows.length; b++)
    {
    columns=split(rows[b],"\t");
    g=parseInt(columns[0]) + 1;
    }
        }
    else
        {
        g = 0;
        }
// CORE
for (i = g ; i < images.length; i++)
{
    inputPath = inImage + images[i];
    n = split(images[i],".");
    name = n[0];
    open(inputPath);
// Number of extractions
Dialog.create("Rosette number");
Dialog.addNumber("How many rosette do you want to extract? ", 1);
Dialog.show();
numberOfIteration = Dialog.getNumber();
    for (k = 1 ; k <= numberOfIteration; k++)
    {
    Dialog.create("Plant name");
    Dialog.addString("Plant name", 1);
    Dialog.show();
    p = Dialog.getString();
    Dialog.create("Leaf number");
    Dialog.addNumber("How many leaves do you want to extract? ", 1);
    Dialog.show();
    numberLeaves= Dialog.getNumber();
    Dialog.create("Leaf number starting");
    Dialog.addNumber("Number of the starting leaf ", 1);
    Dialog.show();
    startingnumberLeaves= Dialog.getNumber();
        for (j = startingnumberLeaves ; j<= numberLeaves; j++)
        {
        run("Wand Tool...", "mode=Legacy tolerance=25");
        selectionString = "Select leaf, then press Ok.";
        waitForUser(selectionString);
        run("Duplicate...", "title=selection.JPG");
        run("Clear Outside");
        saveAs("png" , outImage + "Plant_" + p + "_Leaf_" + j ) ;
        close();
        }
    }
Dialog.create("");
Dialog.addChoice("", newArray("continue", "abort"));
Dialog.show();
process = Dialog.getChoice();
if (process == "abort")
{
    print(i);
    selectWindow("Log");
        saveAs("text", outImage + "Log.txt");
    run("Close");
    run("Close");
    exit();
}
else
{
    print(i);
    selectWindow("Log");
        saveAs("text", outImage + "Log.txt");
    run("Close");
    run("Close");
}}
```

R-Script

After all pictures of single leaves are cleaned and named properly with the *ImageJ* script, the second part of this tool will then read in the cleaned files and quantify the fraction of differently coloured pixels (green, green-yellow, yellow, brown, purple). This *R* script also includes several functions to summarize, analyse, and present the gained data (used *R* version: 3.3.1 (2016-06-21)). Furthermore, instances are implemented to give the user the possibility to check his data for integrity, as for example single leaf pixel analysis to identify problematic images with high levels of *unknown* pixels etc.

Definition of colours

ACA is based on classification of each pixel of a picture depending on its *Hue*, *Saturation* and *Value* (HSV) readings. In contrast to the widely used *Red*, *Green*, *Blue* (RGB) colour coding, HSV gives the opportunity to easily identify colours. In an RGB coded image the actual colour of a pixel would have to be calculated via the relative portion of each channel. Whereas in an HSV coded image the H channel defines the observable colourtone (set by different colour wavelengths) and is expressed in an angle from 0 to 360 degrees on a colour wheel. If different levels of saturation are also included, identifying a colour in RGB would be a rather complex task. In HSV, saturation is a measure of colour intensity indicating the range of grey in the colour-space from 0 to 100%. It allows distinguishing two close colours with the same predominant wavelength but with a distinct grey value. A lower saturation level reflects a faded colour, which means the colour contains more grey (S channel). The V channel directly represents the brightness of the colour image from 0 to 1 (i.e., totally black to white) and varies with S (Fig. 2).

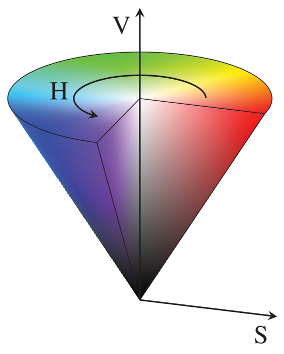
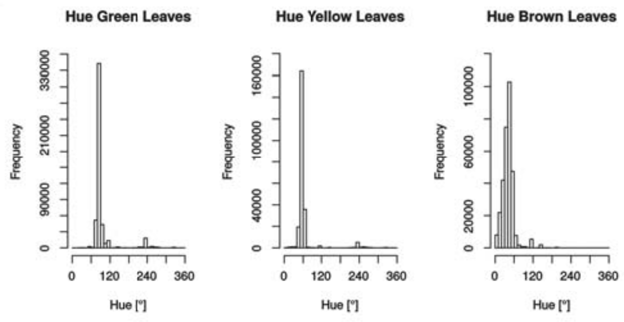


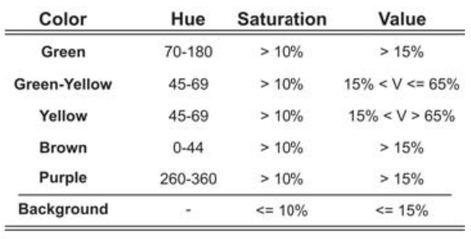
Figure 2: Schematic of Hue-Saturation-Value-colourspace (HSV). *Tikz* drawing source code modified after http://tex.stackexchange.com/.

To get an idea of leaf colours in HSV space, pictures of single leaves were first categorized by eye according to their predominant colouration. Then frequency plots of the respective hue per pixel for each class of pictures were build (Fig. 3A). Here, green as well as yellow pictures show very narrow peaks for hue, while brown exhibits a broader hue-range. This stems from brown leaves with remaining yellow and reddish spots. Nevertheless, conditions for each colour were defined based on the in figure 3A shown frequency plots (Fig. 3B).
The highest peaks for green, yellow and brown were observed at 86°, 56.5° and 43°, respectively. Green pixels were defined to be ranging in hue from 70° to 180°. Yellow was defined to be in the range of H: 45°-69° with a $V > 65\%$. Additionally, dark-greenish tones of yellow were assigned to green-yellow (H: 45-69°, $V \leq 65\%$). Brown H was set to 0°-49°, also including reddish-brown pixels. All identifications exclude background pixels, which are recognized by either a low S (white and black pixels), or a low V (black pixels). Purple was categorized manually by us to H = 260°-360° (magenta/purple). Remaining pixels, which were not identifiable before are classified as *unknown*.

A)



B)



Color	Hue	Saturation	Value
Green	70-180	> 10%	> 15%
Green-Yellow	45-69	> 10%	15% < V <= 65%
Yellow	45-69	> 10%	15% < V > 65%
Brown	0-44	> 10%	> 15%
Purple	260-360	> 10%	> 15%
Background	-	<= 10%	<= 15%

Figure 3: Defined HSV Ranges. A) Frequency plots of *Hue* of pictures of green, yellow and brown leaves (N= 12,18 and 24, respectively). B) Overview table of derived definitions.

Required Packages

Libraries *ggplot2*, *imager*, *tidyr*, *dplyr*, *stringr*, *svglite* and *grid* (Wickham, 2009; Barthelme, 2017; Wickham, 2017*a*; Wickham, 2016 and Wickham, 2017*b*) are required for full functionality of this script. The *ImageR* library should be in version 0.4 or higher. On GNU/Linux systems further system packages are required for this package (*libfftw3-dev*, *libtiff* and *g++* should pull all necessary dependencies).
Additionally, the *multiplot* function needs to be inserted manually via the following code from the *R-Cookbook* (Chang, 2012; Link: *Multiple graphs on one page (ggplot2)*).

```
  ##########################
  ### Multiplot function ###
  ##################################################################################
  ### From Cookbook-R.com (Winston Chang, 2012)                                  ###
  ### http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_%28ggplot2%29/ ###
  ##################################################################################

multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

 if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                             layout.pos.col = matchidx$col))
    }
  }
}
```

Note: Only the *ImageR* library is required for image analysis. Thus, if graphical result presentation is done with a spreadsheet calculator, the remaining libraries are not needed.

Main Script

Prepare Directories and Results-Frame. The working directory must be set to the directory containing the cleaned leaf images. Here, folders are created to save the results as well as the control images (described below) in. Furthermore, in our example a file containing information on plant age is loaded to extract the corresponding data associated with the file name (*IDsexample.txt*, WAS = Weeks After Sowing). Also, the only genotype analysed here is Columbia-0. This is why the genotype is not extracted from the filename or an *ID* file, but assigned once after frame creation. Information can also be extracted from the filename only. E.g. with a filename structure like *'Line Col-0 - Plant 01 - WAS 7 - Leaf 07.png'* the following code would apply:

```
results$Filename[i] <- as.character(files[i])
results$Line[i] <- substr(files[i], 6, regexpr(" -", files[i])[1] - 1)
results$Plant[i] <- substr(files[i], regexpr("Plant ", files[i])[1] + 6, regexpr(" - WAS",
    files[i])[1] - 1)
results$Leaf[i] <- substr(files[i], regexpr("Leaf ", files[i])[1] + 5, regexpr(".png",
    files[i])[1] - 1)
results$WAS[i] <- substr(files[i], regexpr("WAS ", files[i])[1] + 4, regexpr(" - Leaf",
    files[i])[1])
```

Colour Identification. The colour identification loop then successively loads each PNG image available in the folder specified before. After loading, the image is converted to HSV colour-space, file information is extracted (in the here presented example, filename structure is: *Plant_X_Leaf_Y.png*) and each pixel is assigned to a temporary vector according to its colour, then the information is put into the results frame. ***Control Image Output.*** All the defined HSV ranges have been tested to work properly for *Arabidopsis thaliana* plants grown under multiple different conditions in our chambers. Still, adjustments for different species and/or growth conditions may be required. To be able to adjust to differing conditions, the script includes an *control image output*.
Here, the corresponding pixels are oversaturated (displayed red in the output image) and modified images are saved in PNG format (Fig. 4). This part is not needed for results calculation but only for visualisation and troubleshooting. For this reason, image output may be disabled by simply closing the loop before with *}*, thus considerably reducing required computing time.

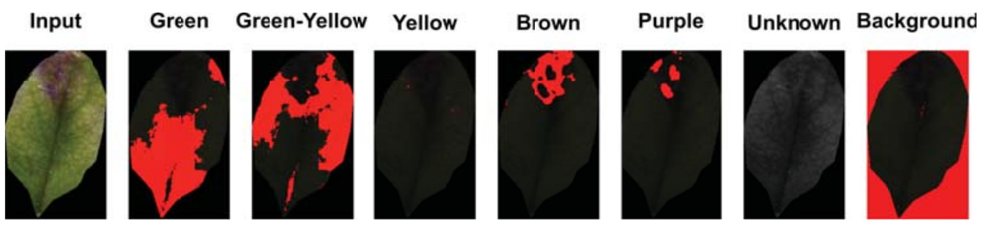


Figure 4: Example Output of Control Images. From left to right: Input image, pixels recognized as green, green-yellow, yellow, brown, purple, unknown and background. No unknown pixels are recognized, thus no markings are visible.

```r
  ##################################
  ### Main ACA Colour ID Script ###
  ##################################


  ##########################
  ### Required Packages ###
  ##########################

  library(ggplot2)
  library(svglite)
  library(imager)
  library(tidyr)
  library(dplyr)
  library(stringr)
  library(grid)

  ############################################
  ### Prepare Directory and Results Frame ###
  ############################################

setwd('~/Documents/### Colorimetrie Analysis 01/markdown example')
dir.create(file.path(paste(getwd(),'/Results',sep='')))
dir.create(file.path(paste(getwd(),'/Results/Control Images',sep='')))

files <- list.files(path =getwd(),pattern=".png")
count <- as.numeric(length(files))
results <- data.frame(Line = character(count),
                      Plant = numeric(count),
                      Leaf = numeric(count),
                      WAS = numeric(count),
                      Total = numeric(count),
                      BG = numeric(count),
                      WOBG= numeric(count),
                      GreenYellow = numeric(count),
                      PercentGreenYellow = numeric(count),
                      Green = numeric(count),
                      PercentGreen = numeric(count),
                      Yellow = numeric(count),
                      PercentYellow =numeric(count),
                      Brown=numeric(count),
                      YB = numeric(count),
                      PercentBrown = numeric(count),
                      Unknown = numeric(count),
                      PercentUnknown = numeric(count),
                      Purple = numeric(count),
                      PercentPurple = numeric(count),
                      Filename = character(count),
                      stringsAsFactors = FALSE)

 IDs <- read.delim("~/Documents/### Colorimetrie Analysis 01/IDsexample.txt",
                  header =TRUE, sep = "\t", dec=".", stringsAsFactors = FALSE)
 results$Line <- "Col-0"


  #############################
  ### Colour Identification ###
  #############################

for (i in 1:length(files)) {
  imagerHSV <- load.image(files[i])
  imagerHSV <- RGBtoHSV(imagerHSV)

  ##################################################
  ### Insert your filename processing below here ###
  ##################################################

  results$Filename[i] <- as.character(files[i])
  results$Plant[i] <- substr(files[i],7,regexpr("_L",files[i])[1]-1)
  results$Leaf[i] <- as.numeric(substr(files[i],regexpr("f_",files[i])[1]+2,
                                       regexpr(".png",files[i])[1]-1))
  results$WAS[i] <- IDs$Week[which(IDs$idPot == results$Plant[i])[1]]


  ##########################
  ### Colour recognition ###
  ##########################

  tempBG <- which(imagerHSV[,,1,2] <= 0.1|imagerHSV[,,1,3] <= 0.15)
  tempGY <- which((is.element(round(imagerHSV[,,1,1]), 45:69) &
                   imagerHSV[,,1,3] <= 0.65) &
                  (imagerHSV[,,1,2] > 0.1 & imagerHSV[,,1,3] > 0.15))
  tempDG <- which(is.element(round(imagerHSV[,,1,1]), 70:180) &
                   (imagerHSV[,,1,2] > 0.1 & imagerHSV[,,1,3] > 0.15))
  tempY <- which((is.element(round(imagerHSV[,,1,1]), 45:69) &
                   imagerHSV[,,1,3] > 0.65) &
                  (imagerHSV[,,1,2] > 0.1 & imagerHSV[,,1,3] > 0.15))
  tempB <- which(is.element(round(imagerHSV[,,1,1]), 0:44) &
                  (imagerHSV[,,1,2] > 0.1 & imagerHSV[,,1,3] > 0.15))
  tempP <- which(is.element(round(imagerHSV[,,1,1]), 260:360)&
                  (imagerHSV[,,1,2] > 0.1 & imagerHSV[,,1,3] > 0.15))
```

```r
tempall <- which(is.element(round(imagerHSV[,,1,1]),0:360))
tempallcolors <- c(tempBG, tempB, tempDG, tempGY, tempY, tempP)
tempunknown <- which(!is.element(tempall, tempallcolors))

results$Total[i] <- (nrow(imagerHSV)*ncol(imagerHSV))
results$BG[i] <- length(tempBG)
results$GreenYellow[i] <- length(tempGY)
results$Green[i] <- length(tempDG)
results$Yellow[i] <- length(tempY)
results$Brown[i] <- length(tempB)
results$Unknown[i] <- length(tempunknown)
results$Purple[i] <- length(tempP)
results$WOBG[i] <- length(tempall) - length(tempBG)


################################
### Control image output.    ###
### If not used close loop   ###
### above with "}"           ###
################################

unknown <- HSVtoRGB(imagerHSV)
unknown[tempBG] <- 5
save.image(im = unknown, file = paste(getwd(), "/Results/Control Images/",
                                      substr(files[i],1,regexpr(".png",
                                      files[i])[1]-1),"background.png", sep=""))
unknown <- HSVtoRGB(imagerHSV)
unknown[tempGY] <- 5
save.image(im = unknown, file = paste(getwd(), "/Results/Control Images/",
                                      substr(files[i],1,regexpr(".png",
                                      files[i])[1]-1),"greenyellow.png", sep=""))
unknown <-HSVtoRGB(imagerHSV)
unknown[tempY] <- 5
save.image(im = unknown, file = paste(getwd(), "/Results/Control Images/",
                                      substr(files[i],1,regexpr(".png",
                                      files[i])[1]-1),"yellow.png", sep=""))
unknown <- HSVtoRGB(imagerHSV)
unknown[tempB] <- 5
save.image(im = unknown, file = paste(getwd(), "/Results/Control Images/",
                                      substr(files[i],1,regexpr(".png",
                                      files[i])[1]-1),"brown.png", sep=""))
unknown <- HSVtoRGB(imagerHSV)
unknown[tempDG] <- 5
save.image(im = unknown, file = paste(getwd(), "/Results/Control Images/",
                                      substr(files[i],1,regexpr(".png",
                                      files[i])[1]-1),"green.png", sep=""))
```

```
  unknown <- HSVtoRGB(imagerHSV)
  unknown[tempP] <- 5
  save.image(im = unknown, file = paste(getwd(), "/Results/Control Images/",
                                       substr(files[i],1,regexpr(".png",
                                       files[i])[1]-1),"purple.png", sep=""))
  unknown <- HSVtoRGB(imagerHSV)
  unknown[tempunknown] <- 5
  save.image(im = unknown, file = paste(getwd(), "/Results/Control Images/",
                                       substr(files[i],1,regexpr(".png",
                                       files[i])[1]-1),"unknown.png", sep=""))
}
```

Results Output and Processing

Raw Data Export and Preparation for later Processing. In the following, percentages are calculated and raw results are saved in a comma separated file which can be imported into any spreadsheet calculation program (see part A in the script below). Percentages are calculated after subtraction of background pixels from the total number of pixels, but still including *unknown* pixels. Furthermore, results are put into *long-table* format for easier processing via *gather*, summary-statistics are calculated with *ddply* and finally the order of the colours is defined (part B in the script below). Definition of the order of colours is necessary for later plotting of stacked barcharts. In the example given, the order is from top to bottom: *unknown*, *purple*, *brown*, *yellow*, *green-yellow* and *green* pixels.

```
  ##############   A   ##############
  ##################################
  ### Calculation of percentages ###
  ### and output of csv file     ###
  ##################################

  results$PercentGreen <- as.numeric(results$Green/(results$WOBG/100))
  results$PercentGreenYellow <- as.numeric(results$GreenYellow/(results$WOBG/100))
  results$PercentYellow <- as.numeric(results$Yellow/(results$WOBG/100))
  results$PercentBrown <- as.numeric(results$Brown/(results$WOBG/100))
  results$PercentUnknown <- as.numeric(results$Unknown/(results$WOBG/100))
  results$PercentPurple <- as.numeric(results$Purple/(results$WOBG/100))
  results$Line <- as.factor(results$Line)

  write.table(results, file = paste(getwd(), "/Results/rawresults.txt",sep=""),
              quote = FALSE, row.names = FALSE)


  ###################   B   ###################
  #############################################
  ### Data preparation for later processing ###
  #############################################

  gathereddata <- results %>% gather(key="color",value="Percent",
                                   c(PercentGreen,PercentGreenYellow,
                                     PercentYellow,PercentBrown,
                                     PercentUnknown,PercentPurple), na.rm=TRUE)
  gathereddata$color <- str_split_fixed(string = gathereddata$color,
                                      pattern = "Percent", n=2)[,2]
  gathereddata$Line <- as.factor(gathereddata$Line)


  summarydatawholeplant <- ddply(gathereddata, c("Line", "WAS", "color"), summarise,
                               N = length(Percent),
                               mean = mean(Percent),
                               sd = sd(Percent),
                               se = sd / sqrt(N),
                               med = median(Percent),
                               inter = IQR(Percent),
                               lower = quantile(Percent)[2],
                               upper = quantile(Percent)[4],
                               ybegin = mean - se,
                               yend =  mean + se
                                )

  summarydatasingleleaf <- ddply(gathereddata, c("Line", "Leaf", "WAS", "color"),summarise,
                               N = length(Percent),
                               mean = mean(Percent),
                               sd = sd(Percent),
                               se = sd / sqrt(N),
                               med = median(Percent),
                               inter = IQR(Percent),
                               lower = quantile(Percent)[2],
                               upper = quantile(Percent)[4],
                               ybegin = mean - se,
                               yend =  mean + se
                               )

  summarydatasingleleaf$Line <- as.factor(summarydatasingleleaf$Line)
  summarydatawholeplant$Line <- as.factor(summarydatawholeplant$Line)
  summarydatawholeplant$WAS <- as.factor(summarydatawholeplant$WAS)


  summarydatasingleleaf$color <- factor(summarydatasingleleaf$color, levels =
                                        c("Unknown","Purple","Brown",
                                          "Yellow","GreenYellow","Green"))
  summarydatawholeplant$color <- factor(summarydatawholeplant$color, levels =
                                        c("Unknown","Purple","Brown",
                                          "Yellow","GreenYellow","Green"))
  gathereddata$color <- factor(gathereddata$color, levels =
                               c("Unknown","Purple","Brown","Yellow",
                                 "GreenYellow","Green"))
```

General Information. To get an impression on meaningfulness of single leaf analysis, the number of leaves analysed per position is plotted per genotype over time (Fig. 5A). The following for-loop first generates plots which summarize pixels identified per harvest and genotype (Fig. 5B). These plots should always be inspected, especially for *unknown* pixels. In the second set, each colour is plotted over time per genotype and *leaf position*. In the case of high percentages of *unknown* pixels observed in B, this set of plots allows the user to decide whether this problem arises from single pictures or if it is a more general problem of colour identification (Fig. 5C).

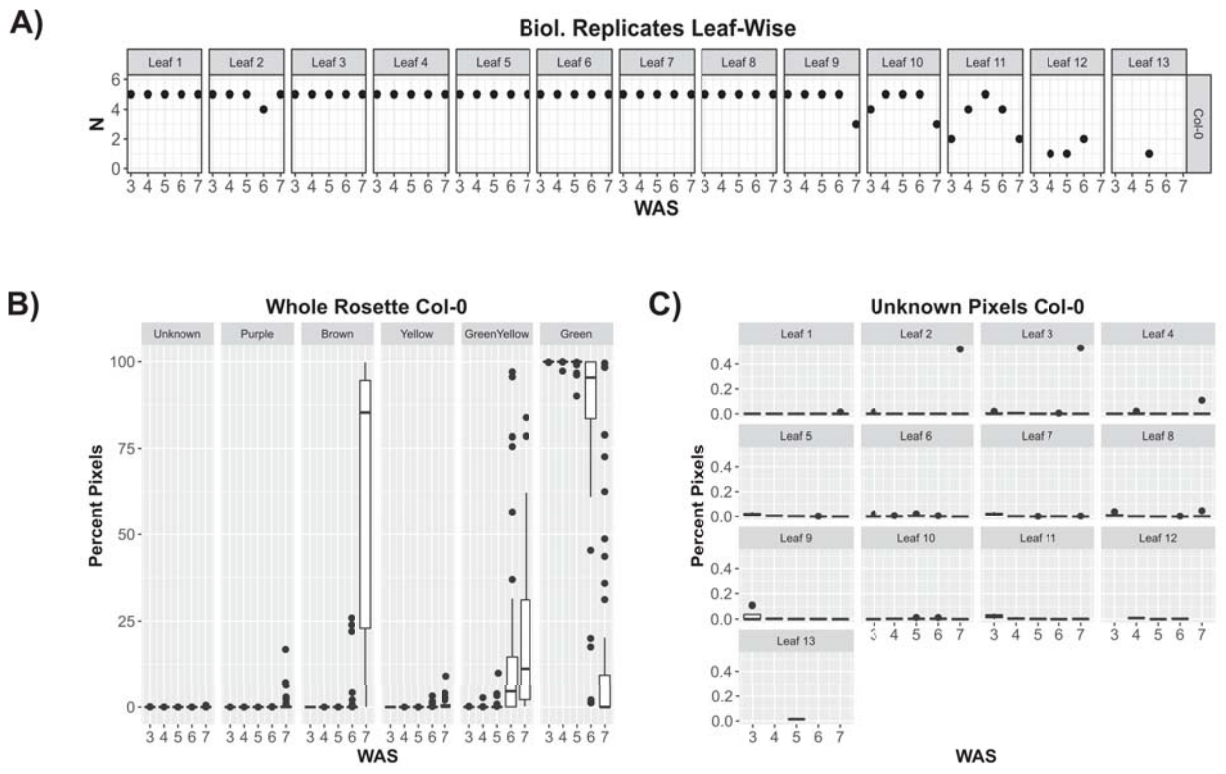


Figure 5: General Information Plots. A) Number (N) of biological replicates analysed per leaf position and genotype over weeks after sowing. B) Percentage of each colour on whole rosette level over weeks after sowing. C) Percentage of pixels *unknown* per leaf position over time. WAS = Weeks after Sowing.

Stacked Barcharts. For the following plots data summarized with *ddply* is used. Because stacked bar charts are generated, the errorbar coordinates are re-calculated as the sum of all colours plotted below the corresponding colour ± SE (e.g. with *unknown* on top, its errorbar y-coordinates are the summed means of all other colours ± its own SE). Furthermore, to have the bar in only one direction, *'yend'* is set to the corresponding mean value. The function uses *multiplot* to generate multiple stacked bar charts in one figure per genotype. First each leaf position is plotted, the last plot summarizes the whole rosette colour. These plots are generated by genotype and exported into *SVG* format (Fig. 6A).
The same information is presented in the next set plots, but here the genotypes are not plotted separately but next to each other. Thus making direct leaf wise comparisons easier. Depending on the number of leaves analysed, this plot tends to get very big. This is why the leaf positions to be used for this plot have to be specified before (*minleaf*, *maxleaf*). Nevertheless, if needed all positions analysed may be plotted (Fig. 6B).

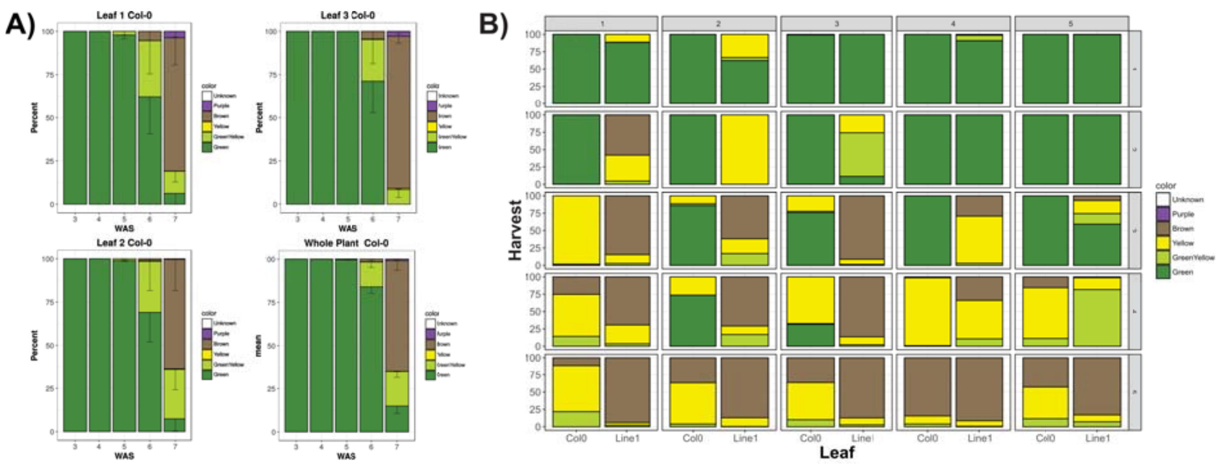


Figure 6: Stacked Barcharts. A) Example plot for stacked barcharts per genotype. Only the first three leaves are shown here for better illustration. B) Exemplary plot for leaf-wise genotype comparison Col-0 vs. a delayed line (*Line1*, unpublished data). Leaves 1-5 were chosen, no error-bars given as only one replicate was analysed.

```
  ##########################
  ### Plotting Functions ###
  ##########################



  ####################################
  ### Plots on general information ###
  ####################################



text.on.each.panel <-"Leaf "

ggplot(data= summarydatasingleleaf, aes(x=WAS, y=N))+
  geom_count(stat="identity")+
  theme_bw()+
  theme(plot.title = element_text(hjust = 0.5,size=16,face="bold"),
        axis.text=element_text(size=12),
        axis.title=element_text(size=14,face="bold")) +
  scale_y_continuous(limits = c(0,(max(summarydatasingleleaf$N)+1))) +
  ggtitle("Biol. Replicates Leaf-Wise") +
  facet_grid(Line~Leaf, labeller =
               label_bquote(cols=paste(.(text.on.each.panel),.(Leaf), sep="")))
ggsave(paste(getwd(),"/Results/Replicates.svg",sep=""), plot=last_plot(),
       width=0.6*max(summarydatasingleleaf$Leaf)+0.5*max(summarydatasingleleaf$WAS),
       height=2*length(levels(summarydatasingleleaf$Line)))

attach(gathereddata)
for (k in 1:length(levels(Line))){
  ggplot(data = subset(gathereddata, Line == levels(Line)[k]),
         aes(x=WAS, y=Percent,group=WAS))+
    geom_boxplot(stat="boxplot")+
    labs(y="Percent Pixels")+
    ggtitle(paste("Whole Rosette ", levels(Line)[k]))+
    theme(plot.title = element_text(hjust = 0.5,size=16,face="bold"),
          axis.text=element_text(size=12),
          axis.title=element_text(size=14,face="bold")) +
    facet_wrap(~color, nrow = ceiling(length(levels(as.factor(Line)))/2))
  ggsave(paste(getwd(),"/Results/", levels(Line)[k],"-Colors_Whole_Rosette.svg", sep=""),
         plot=last_plot())

  for (j in 1:length(levels(color))){
  ggplot(data = subset(gathereddata, Line == levels(Line)[k] & color == levels(color)[j]),
         aes(x=WAS, y=Percent,group=WAS))+
    geom_boxplot(stat="boxplot")+
    labs(y="Percent Pixels")+
    ggtitle(paste(levels(color)[j], "Pixels" , levels(Line)[k]))+
    theme(plot.title = element_text(hjust = 0.5,size=16,face="bold"),
          axis.text=element_text(size=12),
          axis.title=element_text(size=14,face="bold")) +
    facet_wrap(~Leaf, , labeller =
                 label_bquote(cols=paste(.(text.on.each.panel),.(Leaf), sep="")))
  ggsave(paste(getwd(),"/Results/",levels(Line)[k],"-",levels(color)[j],
               "_Pixels_PerLeaf.svg", sep=""), plot=last_plot())
  }
}
detach(gathereddata)

  ###################################
  ### Adjust errorbar coordinates ###
  ### for stacked barcharts       ###
  ###################################

frames <- c("summarydatasingleleaf", "summarydatawholeplant")
for (i in 1:2){
  tf <- get(frames[i])
    tf$ybegin[tf$color=="Unknown"] <- tf$ybegin[tf$color=="Unknown"] +
                                  tf$mean[tf$color=="Yellow"] +
                                  tf$mean[tf$color=="Green"] +
                                  tf$mean[tf$color=="GreenYellow"] +
                                  tf$mean[tf$color=="Purple"] +
                                  tf$mean[tf$color=="Brown"]
    tf$yend[tf$color=="Unknown"] <- tf$mean[tf$color=="Unknown"] +
                                tf$mean[tf$color=="Yellow"] +
                                tf$mean[tf$color=="Green"] +
```

```r
                                tf$mean[tf$color=="GreenYellow"] +
                                tf$mean[tf$color=="Purple"] +
                                tf$mean[tf$color=="Brown"]
    tf$ybegin[tf$color=="Purple"] <- tf$ybegin[tf$color=="Purple"] +
                                 tf$mean[tf$color=="Yellow"] +
                                 tf$mean[tf$color=="Brown"] +
                                 tf$mean[tf$color=="Green"] +
                                 tf$mean[tf$color=="GreenYellow"]
    tf$yend[tf$color=="Purple"] <- tf$mean[tf$color=="Purple"] +
                               tf$mean[tf$color=="Yellow"] +
                               tf$mean[tf$color=="Green"] +
                               tf$mean[tf$color=="GreenYellow"] +
                               tf$mean[tf$color=="Brown"]
    tf$ybegin[tf$color=="Brown"] <- tf$ybegin[tf$color=="Brown"] +
                                tf$mean[tf$color=="Yellow"] +
                                tf$mean[tf$color=="Green"] +
                                tf$mean[tf$color=="GreenYellow"]
    tf$yend[tf$color=="Brown"] <- tf$mean[tf$color=="Brown"] +
                              tf$mean[tf$color=="Yellow"]+
                              tf$mean[tf$color=="Green"] +
                              tf$mean[tf$color=="GreenYellow"]
    tf$ybegin[tf$color=="Yellow"] <- tf$ybegin[tf$color=="Yellow"]+
                                 tf$mean[tf$color=="Green"] +
                                 tf$mean[tf$color=="GreenYellow"]
    tf$yend[tf$color=="Yellow"] <- tf$mean[tf$color=="Yellow"] +
                               tf$mean[tf$color=="Green"] +
                               tf$mean[tf$color=="GreenYellow"]
    tf$ybegin[tf$color=="GreenYellow"] <- tf$ybegin[tf$color=="GreenYellow"] +
                                     tf$mean[tf$color=="Green"]
    tf$yend[tf$color=="GreenYellow"] <- tf$mean[tf$color=="GreenYellow"] +
                                   tf$mean[tf$color=="Green"]
    tf$ybegin[tf$color=="Green"] <- tf$ybegin[tf$color=="Green"]
    tf$yend[tf$color=="Green"] <- tf$mean[tf$color=="Green"]
    assign(frames[i], tf)
  }


  ##############################
  ### Plot stacked barcharts ###
  ### PER GENOTYPE           ###
  ##############################

for (k in 1:length(levels(summarydatasingleleaf$Line))){
  p <- list()
  for (i in min(summarydatasingleleaf$Leaf[summarydatasingleleaf$Line==
            levels(summarydatasingleleaf$Line)[k]]):
```

```r
            max(summarydatasingleleaf$Leaf[summarydatasingleleaf$Line==
            levels(summarydatasingleleaf$Line)[k]])) {
              leaf <- subset(summarydatasingleleaf, summarydatasingleleaf$Leaf==i &
                            summarydatasingleleaf$Line==
                            levels(summarydatasingleleaf$Line)[k])
      p[[i]] <- (ggplot(leaf, aes(x=WAS, y= mean,fill=color))+
                 geom_bar(stat = "identity",color="black")+
                 scale_fill_manual(values=c("white","purple","burlywood4",
                                            "yellow","green", "chartreuse4"))+
                 theme_bw()+
                 labs(y = "Percent Pixels")+
                 theme(plot.title = element_text(hjust = 0.5,size=16,face="bold"),
                       axis.text=element_text(size=12),
                       axis.title=element_text(size=14,face="bold")) +
                 geom_errorbar(aes(x=WAS, ymax=yend, ymin=ybegin), na.rm=TRUE,
                               size=0.25,width=0.25)+
                 ggtitle(paste("Leaf",i, leaf$Line)))
      }
  wholeplant <- subset(summarydatawholeplant, summarydatawholeplant$Line==
                       levels(summarydatawholeplant$Line)[k])
   p[[i+1]] <- (ggplot(wholeplant,aes(x=WAS, y= mean,fill=color))+
                 geom_bar(stat = "identity",color="black")+
                 scale_fill_manual(values=c("white","purple","burlywood4",
                                            "yellow","green","chartreuse4"))+
                 theme_bw()+
                 theme(plot.title = element_text(hjust = 0.5,size=16,face="bold"),
                       axis.text=element_text(size=12),
                       axis.title=element_text(size=14,face="bold")) +
                 geom_errorbar(aes(x=WAS, ymax=yend, ymin=ybegin), na.rm=TRUE,
                               size=0.25,width=0.25)+
                 ggtitle(paste("Whole Rosette ", leaf$Line)))
  svg(filename = paste(getwd(), "/Results/stacked ",levels(summarydatasingleleaf$Line)[k],
                       ".svg",sep=""),w=5.17,h=(5.28*(max(summarydatasingleleaf$Leaf
                        [summarydatasingleleaf$Line==
                        levels(summarydatasingleleaf$Line)[k]])+1)))
    multiplot(plotlist=p[min(summarydatasingleleaf$Leaf[summarydatasingleleaf$Line==
                            levels(summarydatasingleleaf$Line)[k]]):
                       max(summarydatasingleleaf$Leaf[summarydatasingleleaf$Line==
                            levels(summarydatasingleleaf$Line)[k]]+1)])
  dev.off()
}


  ################################
  ### Plot stacked barcharts ###
  ### ALL GENOTYPES          ###
  ###############################


minleaf <- 1
maxleaf <- 10
ggplot(subset(summarydatasingleleaf, is.element(summarydatasingleleaf$Leaf,
                                                minleaf:maxleaf)),
       aes(x=Line, y= mean,fill=color))+
  geom_bar(stat = "identity",color="black")+
  labs(x="Leaf",y="WAS")+
  scale_fill_manual(values=c("white","purple","burlywood4",
                             "yellow","green", "chartreuse4"))+
  theme_bw()+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=20,face="bold")) +
  geom_errorbar(aes(x=Line, ymax=yend, ymin=ybegin), na.rm=TRUE,
                size=0.25,width=0.25)+
  facet_grid(WAS~Leaf, labeller =
               label_bquote(cols=paste(.(text.on.each.panel),.(Leaf), sep="")))
ggsave(paste(getwd(),"/Results/ColorLeafLineTime.svg",sep=""),
       plot=last_plot(),height = max(summarydatasingleleaf$WAS)*1.5,
       width=((maxleaf-(minleaf-1))*(1.2*length(levels(summarydatasingleleaf$Line))))
       , limitsize=FALSE)
```

References

Barthelme S. 2017. *Imager: Image processing library based on 'cimg'*. https://cran.r-project.org/package=imager.

Chang W. 2012. *Cookbook for r*. www.cookbook-r.com.

R Core Team. 2016. *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing.

Schindelin J, Rueden CT, Hiner MC, Eliceiri KW. 2015. The imagej ecosystem: An open platform for biomedical image analysis. Molecular Reproduction and Development **82**, 518–529.

Wickham H. 2009. *Ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York.

Wickham H. 2016. *Dplyr: A grammar of data manipulation*. https://cran.r-project.org/package=dplyr.

Wickham H. 2017*a*. *Tidyr: Easily tidy data with 'spread()' and 'gather()' functions*. https://cran.r-project.org/package=tidyr.

Wickham H. 2017*b*. *Stringr: Simple, consistent wrappers for common string operations*. https://cran.r-project.org/package=stringr.