/\*\*\*\*\*CALIBRATION MACRO FOR WALNUT PELLICLE COLOR ANALYSIS\*\*\*\*\*

//Revised 18 December 2018 by Peter McAtee (peter.mcatee@plantandfood.co.nz)

// Created 13 April 2017 by Irwin R. Donis-Gonzalez

// Donis-Gonzalez, Irwin, GM Sideli, SM Bergman, DC Slaughter, C, Crisosto. 2020. Color Vision System to Assess English Walnut (*Juglans Regia*) Kernel Pellicle Color. Postharvest Biology and Technology*.*

//

// An ImageJ macro color calibrator developed for walnut pellicle color analysis

//

// The function of this macro is to create illumination uniformity correction images and to develop the calibration constants

// required to convert the sRGB color values from the digital images into CIE L\*, a\*, b\* color values matching standard

// colorimeter measurements.

//

// This version includes the Label Mask generation, spatial scale determination, Image Size Std. Values, and background color determination.

// Images are automatically resized to match the BASLER acA2040-25gc camera with a 14R0034188 lense output Image Size

// Omits any calibration data where the L\* image data exceeds 517, the limit for saturation

// This version also determines the C\* (chroma) and h\* (hue angle) values for calibration.

// NOTE THIS REQUIRES THE CORRECTED Color Transformer plugin.

// The original came from http://rsbweb.nih.gov/ij/plugins/color-transforms.html

// The corrected version must be put into the plugins folder (or a subfolder) in order for the macro to run.

// This plugin converts an RGB color image into a color space for color analysis. The XYZ, Yxy, YUV, YIQ, Luv, Lab, AC1C2,

// I1I2I3, Yuv, YQ1Q2, HSI, HSV, HSL, LCHLuv, LSHLuv and LCHLab color spaces are supported. The lab option is used to

// convert into the CIE L\*, a\*, b\* color space.

// the macro is set to display all images. Change the word "false" to "true" in the following command to switch modes.

// If boolean is true, the interpreter enters batch mode and images are not displayed

setBatchMode(false); // false);

// Set ImageJ binary option to uncheck black background, so that the mask will operate consistently between computers;

run("Options...", "iterations=1 edm=Overwrite count=1");

// The next command forces ImageJ to convert 32-bit images to 8-bit images without UNWANTED scaling;

run("Conversions...", " ");

// clear the log window

print("\\Clear");

// Ask user to identify the folder containing the X-Rite chart and White Balance images

// This version assumes that the X-Rite chart image is called Xrite.JPG and that the white balance image is called WB.JPG

dir1 = getDirectory("Please select the directory containing the X-Rite chart image and the White Balance Image");

//Ask user to identify the folder containing the text file with the Minolta color values for the X-Rite chart

// This version assumes that the text file is titled Minolta\_Lab\_Cal\_Values\_Imagej\_order.txt and follows a standard format

dir2 = getDirectory("Please select the directory that contains the text file of the Minolta Color Values for the X-Rite chart");

//Ask user if they want to ignore the Label Mask // Currently set as not having labels, to avoid confusion as it hasn't been requested for this purpuse, but it can be incorporated in the future.

UseLabel = 0;

//UseLabel = getNumber("Do the images have labels? (0 for no, 1 for full height on left side, or 2 for expand 50% anywhere): ", UseLabel);

//Ask user if they want to ignore the Label Mask // Currently set as not having labels, to avoid confusion as it hasn't been requested for this purpuse, but it can be incorporated in the future

QRCode = 1;

//QRCode = getNumber("Do the images have a QR barcode? (1 for no, 2 for yes): ", QRCode);

//Open the X-rite image

open(dir1 + "Xrite.JPG");

selectWindow("Xrite.JPG");

//Determine the size of the image set Macro originally developed for 1780 x 1780 pixel image

//This assumes that all images are of similar physical dimensions as Label\_Mask image size of 1780x1780

MyWidth = getWidth();

MyHeight = getHeight();

MyXSizeVal = MyWidth/1780; //Size Standardization factors for other image sizes

MyYSizeVal = MyHeight/1780;

SizeFlag = 0; //Set SizeFlag = 0 if image size matches original Canon Rebel Image size

if( (MyWidth != 1780) || (MyHeight != 1780) )

{

SizeFlag = 1;

run("Size...", "width=1780 height=1780 average interpolation=Bilinear"); // Standardize Image Size if it is not 1780 x 1780

}

// ===== FIRST STAGE CROP OF XRITE CARD ===== //

//Use the whte squares to define the limits of the X-rite card (in order to remove any background)

//Image window Xrite\_Temp.JPG used to do this operation

run("Set... ", "zoom=15");

run("Duplicate...", "title=Xrite\_Temp.JPG"); // Make a temporary copy to use in finding chart placement in image

run("Set... ", "zoom=15");

run("Duplicate...", "title=Xrite\_Temp2.JPG"); // Make a temporary copy to use in finding chart placement in image

run("Set... ", "zoom=15");

selectWindow("Xrite\_Temp.JPG");

run("8-bit"); // Convert temporary copy to a grayscale image

run("Median...", "radius=1"); // Smooth to reduce noise

setAutoThreshold("Minimum dark"); // Threshold function to find the white color squares

run("Convert to Mask"); // Make a mask

run("Set Measurements...", "area centroid bounding redirect=None decimal=3");

run("Analyze Particles...", "size=2000-Infinity circularity=0.00-1.00 show=Nothing display exclude clear");

numObs = nResults;

// Set some variables defining the limits that will be overwritten

XTop=9999;

XLeft=9999;

XBot=-1;

XRight=-1;

for (ix=0; ix<numObs; ix+=1) // Search through results to find the ....

{

BX=getResult("BX", ix); // X coordinate of white square in XRITE chart

BY=getResult("BY", ix); // Y coordinate of white square in XRITE chart

Width=getResult("Width", ix); // Width of white square in XRITE chart

Height=getResult("Height", ix); // Height of white square in XRITE chart

XWidth = BX + Width; // Right side of square

XHeight = BY + Height; // Bottom of square

if(XWidth > XRight) XRight=XWidth; // Calculate the right most edge

if(BX < XLeft) XLeft=BX; // Calculate the left most edge

if(BY < XTop) XTop=BY; // Calculate the top most edge

if(XHeight > XBot) XBot=XHeight; // Calculate the bottom most edge

}

selectWindow("Xrite\_Temp.JPG");

close();

//Use the co-ordinates to select the X-Rite card in the image using a crop box

selectWindow("Xrite.JPG");

XWidth1 = XRight - XLeft + 100;

XHeight1 = XBot - XTop + 200;

XLeft1 = XLeft - 50;

XTop1 = XTop - 50;

selectWindow("Xrite.JPG"); // Go back to color version of chart

makeRectangle(XLeft1, XTop1, XWidth1, XHeight1); // Make rectangle generously enclosing the X-Rite chart

run("Crop");

// ===== CHARACTERISE THE BACKGROUND COLOUR ===== //

run("Split Channels");

// Split the image into the Red, Green and Blue channels

selectWindow("Xrite.JPG (blue)"); // Adjust orientation and size of X-Rite chart in order to extract color of squares

run("Set... ", "zoom=15");

setThreshold(165, 255); // Get white squares in chart

makeRectangle(0, 0, 400, 400); // Get top left corner of chart

run("Set Measurements...", "area mean standard min centroid bounding redirect=None decimal=3");

run("Analyze Particles...", "size=2000-Infinity circularity=0.00-1.00 show=Nothing display exclude clear");

BXTL=getResult("BX", 0); //X coordinate of top left corner of top left white square in XRITE chart

BYTL=getResult("BY", 0); //Y coordinate of top left corner of top left white square in XRITE chart

// Save pixel dimensions of the top left white square for future spatial calibration.

SquareWidth=getResult("Width", 0); //Width of top left white square in XRITE chart

SquareHeight=getResult("Height", 0); //Height of top left white square in XRITE chart

// The squares in the X-Rite chart are about 14mm x 14mm in size.

mmWidth = 14;

mmHeight = 14;

//Deterine the color of the background for future use in produce analysis.

//This selects a area outside the X-Rite card ie the background tray

selectWindow("Xrite\_Temp2.JPG");

makeRectangle(XLeft1 +BXTL, XTop +BYTL -SquareHeight\*4 , SquareWidth\*17.8, SquareHeight); // Get top left corner of chart

run("Crop");

run("Gaussian Blur...", "sigma=5");

run("Split Channels");

wait(20);

//Calculate mean of the background for each colour channel

BcolourArray = newArray("Xrite\_Temp2.JPG (red)","Xrite\_Temp2.JPG (green)","Xrite\_Temp2.JPG (blue)");

for(iMean = 0; iMean < BcolourArray.length; iMean++)

{

// Calculate mean of each colour channel

selectWindow(BcolourArray[iMean]);

run("Clear Results");

run("Set Measurements...", "mean standard redirect=None decimal=3");

run("Measure");

if(iMean == 0)

{

BRedMean=getResult("Mean", 0); // Mean Red of background above XRITE chart

BRedStd=getResult("StdDev", 0); // Std. of Red of background above XRITE chart

}

if(iMean == 1)

{

BGreenMean=getResult("Mean", 0); // Mean Green of background above XRITE chart

BGreenStd=getResult("StdDev", 0); // Std. of Green of background above XRITE chart

}

if(iMean == 2)

{

BBlueMean=getResult("Mean", 0); // Mean Blue of background above XRITE chart

BBlueStd=getResult("StdDev", 0); // Std. of Blue of background above XRITE chart

}

close();

}

// Determine the underlying backgound colour of the image

if( (BBlueMean - BBlueStd\*1.8 > BGreenMean + BGreenStd\*1.8) && (BBlueMean - BBlueStd\*1.8 > BRedMean + BRedStd\*1.8) )

{

BackgroundColor = "Blue";

} else if ( (BRedMean - BRedStd\*1.8 > BGreenMean + BGreenStd\*1.8) && (BRedMean - BRedStd\*1.8 > BBlueMean + BBlueStd\*1.8) )

{

BackgroundColor = "Red";

} else

if ( (BGreenMean - BGreenStd\*1.8 > BRedMean + BRedStd\*1.8) && (BGreenMean - BGreenStd\*1.8 > BBlueMean + BBlueStd\*1.8) )

{

BackgroundColor = "Green";

} else

{

BackgroundColor = "Black";

}

// ===== REALIGNMENT OF XRITE CARD TO COMPENSATE FOR ROTATION OFFSET ===== //

//Use the blue channel to calculate the rotation of the XRITE card and

selectWindow("Xrite.JPG (blue)");

Width = XWidth1 - 401;

makeRectangle(Width, 0, 400, 400); // Get top right corner of chart

run("Set Measurements...", "area mean standard min centroid bounding redirect=None decimal=3");

run("Analyze Particles...", "size=2000-Infinity circularity=0.00-1.00 show=Nothing display exclude clear");

BXTR=getResult("BX", 0); //X coordinate of top left corner of top right white square in XRITE chart

BYTR=getResult("BY", 0); //Y coordinate of top left corner of top right white square in XRITE chart

Degree2 = 180.0 \* ((BYTL - BYTR) / (BXTR - BXTL)) / 3.1416; // rotation angle to make chart "square" with window

//Rotate each colour channel

rotArray = newArray("Xrite.JPG (red)","Xrite.JPG (green)","Xrite.JPG (blue)");

for(iRot = 0; iRot < rotArray.length; iRot++)

{

run("Select None");

selectWindow(rotArray[iRot]);

run("Set... ", "zoom=15");

run("Arbitrarily...", "angle="+Degree2+" grid=1 interpolation=Bilinear");

}

// ===== DO SECOND STAGE CROP OF XRITE CARD ===== //

//Do a second stage crop of the XRITE card using the the Top-Right, Top-Left, Bottom-Right, and Bottom Left white squares

selectWindow("Xrite.JPG (blue)");

// ==== TOP LEFT ==== //

makeRectangle(0, 0, 400, 400); // Get top left corner of re-oriented chart

run("Set Measurements...", "area mean standard min centroid bounding redirect=None decimal=3");

run("Analyze Particles...", "size=2000-Infinity circularity=0.00-1.00 show=Nothing display exclude clear");

BXTL=getResult("BX", 0); //X coordinate of top left corner of top left white square in XRITE chart

BYTL=getResult("BY", 0); //Y coordinate of top left corner of top left white square in XRITE chart

// ==== BOTTOM LEFT ==== //

Height = XHeight1 - 501;

makeRectangle(0, Height, 400, 400); // Get bottom left corner of re-oriented chart

run("Analyze Particles...", "size=2000-Infinity circularity=0.00-1.00 show=Nothing display exclude clear");

numObs = nResults;

XLeft=9999;

XBot=-1;

for (ix=0; ix<numObs; ix+=1)

{

BXBL=getResult("BX", ix); //X coordinate of white square in XRITE chart

BYBL=getResult("BY", ix); //Y coordinate of white square in XRITE chart

Height=getResult("Height", ix); //Height of white square in XRITE chart

XHBL = BYBL + Height;

if(BXBL < XLeft) XLeft=BXBL;

if(XHBL > XBot) XBot=XHBL;

}

BXBL=XLeft; //X coordinate of top left corner of bottom left white square in XRITE chart

BYBL=XBot -Height; //Y coordinate of top left corner of bottom left white square in XRITE chart

// ==== TOP RIGHT ==== //

makeRectangle(Width, 0, 400, 400); // Get top right corner of chart

run("Analyze Particles...", "size=2000-Infinity circularity=0.00-1.00 show=Nothing display exclude clear");

numObs = nResults;

XTop=9999;

XRight=-1;

for (ix=0; ix<numObs; ix+=1)

{

BXTR=getResult("BX", ix); //X coordinate of white square in XRITE chart

BYTR=getResult("BY", ix); //Y coordinate of white square in XRITE chart

Width=getResult("Width", ix); //Width of white square in XRITE chart

XWTR = BXTR + Width;

if(XWTR > XRight) XRight=XWTR;

if(BYTR < XTop) XTop=BYTR;

}

BXTR=XRight; //X coordinate of top left corner of top right white square in XRITE chart

BYTR=XTop; //Y coordinate of top left corner of top right white square in XRITE chart

// ==== BOTTOM RIGHT ==== //

Width = XWidth1 - 401;

Height = XHeight1 - 501;

makeRectangle(Width, Height, 400, 400); // Get bottom right corner of re-oriented chart

run("Analyze Particles...", "size=2000-Infinity circularity=0.00-1.00 show=Nothing display exclude clear");

numObs = nResults;

XBot=-1;

XRight=-1;

for (ix=0; ix<numObs; ix+=1)

{

BXBR=getResult("BX", ix); //X coordinate of white square in XRITE chart

BYBR=getResult("BY", ix); //Y coordinate of white square in XRITE chart

Height=getResult("Height", ix); //Height of white square in XRITE chart

Width=getResult("Width", ix); //Width of white square in XRITE chart

XWBR = BXBR + Width;

XHBR = BYBR + Height;

if(XWBR > XRight) XRight=XWBR;

if(XHBR > XBot) XBot=XHBR;

}

BXBR=XRight; //X coordinate of top left corner of top right white square in XRITE chart

BYBR=XBot - Height; //Y coordinate of top left corner of top right white square in XRITE chart

run("Select None");

// ==== CROP VALUES ==== //

CropLeft = minOf(BXTL, BXBL); // Make more exact rectangle enclosing the color squares in the chart.

CropTop = minOf(BYTL, BYTR);

CropWidth = maxOf(BXTR, BXBR) - CropLeft ;

CropHeight = maxOf(BYBL, BYBR)- CropTop+Height; //Add ~90 pixels to get bottom side

//Use calculated co-ordinates to find the scale bar in order to scale the image

run("Duplicate...", "title=Scale\_Temp.JPG"); // Make a temporary copy to use in finding chart placement in image

selectWindow("Scale\_Temp.JPG");

run("Set... ", "zoom=15");

ScaleWidth = 0.27\* CropWidth;

ScaleHeight = 0.033 \* CropHeight;

makeRectangle(CropLeft +CropWidth\*0.98 -ScaleWidth, CropTop +CropHeight\*1.02 , ScaleWidth, ScaleHeight);

run("Crop");

run("Select None");

run("Median...", "radius=0.5");

run("Sharpen");

setAutoThreshold("Minimum dark");

run("Convert to Mask");

run("Set Measurements...", "area centroid bounding redirect=None decimal=3");

run("Analyze Particles...", "size=30-Infinity circularity=0.00-1.00 show=Nothing display clear");

numObs = nResults;

XscaleLeft = 99999;

XscaleRight = -1;

for (ix=0; ix<numObs; ix+=1)

{

SXBR=getResult("BX", ix); //X coordinate of white scale line in XRITE chart

SYBR=getResult("BY", ix); //Y coordinate of white scale line in XRITE chart

SHeight=getResult("Height", ix); //Height of white scale line in XRITE chart

SWidth=getResult("Width", ix); //Width of white scale line in XRITE chart

if((SWidth < 7) && (SHeight >15)) // Increased line width due to misalignment

{

if(XscaleLeft > SXBR) XscaleLeft = SXBR;

if(XscaleRight < SXBR) XscaleRight = SXBR;

}

}

ScalePixelWidth = XscaleRight - XscaleLeft;

mmScale = 60;

selectWindow("Scale\_Temp.JPG");

close();

//Finally use the calculated crop values to crop each colour channel or a second time

selectWindow("Xrite.JPG (red)");

makeRectangle(CropLeft, CropTop, CropWidth, CropHeight);

run("Crop");

run("Gaussian Blur...", "sigma=3.5"); // Blur image to reduce noise in center of squares

selectWindow("Xrite.JPG (green)");

makeRectangle(CropLeft, CropTop, CropWidth, CropHeight);

run("Crop");

run("Gaussian Blur...", "sigma=3.5");

// Blur image to reduce noise in center of squares

selectWindow("Xrite.JPG (blue)");

makeRectangle(CropLeft, CropTop, CropWidth, CropHeight);

run("Crop");

run("Gaussian Blur...", "sigma=3.5"); // Blur image to reduce noise in center of squares

// ===== OPEN WHITE BALANCE IMAGE ===== //

open(dir1 + "WB.JPG");

selectWindow("WB.JPG");

if( SizeFlag ) run("Size...", "width=1780 height=1780 average interpolation=Bilinear"); // Standardize Image Size

run("Select None");

run("Size...", "width=890 height=890 constrain interpolation=Bilinear"); // Resize to speed up processing

run("Median...", "radius=2"); // Smooth image

WHeight=getHeight();

WWidth=getWidth();

run("Split Channels");

wait(20);

// ===== GENERATE UNIFORM IMAGES TO ADJUST OF ASSEMETRCAL LIGHTING ===== //

//Find the min and max value for each channel in the white balance image

//Colour channels in the following arrays have to be in the same order in this case 1) Red, 2) Green, 3) Blue

WBcolourArray = newArray("WB.JPG (red)","WB.JPG (green)","WB.JPG (blue)");

for(iMin = 0; iMin < WBcolourArray.length; iMin++)

{

// Calculate min and max of each colour channel in teh white balance image

selectWindow(WBcolourArray[iMin]);

run("Select None");

run("Clear Results");

run("Set Measurements...", "min redirect=None decimal=3");

run("Measure");

if(iMin == 0)

{

WB\_Red\_Min=getResult("Min", 0); // Get minimum gray level of white balance chart

WB\_Red\_Max=getResult("Max", 0); // Get maximum gray level of white balance chart

}

if(iMin == 1)

{

WB\_Green\_Min=getResult("Min", 0); // Get minimum gray level of white balance chart

WB\_Green\_Max=getResult("Max", 0); // Get maximum gray level of white balance chart

}

if(iMin == 2)

{

WB\_Blue\_Min=getResult("Min", 0); // Get minimum gray level of white balance chart

WB\_Blue\_Max=getResult("Max", 0); // Get maximum gray level of white balance chart

}

}

// USE THE MAX VALUES TO GENERATE A UNIFORM BACKGROUND FOR EACH COLOUR CHANNEL

WB\_ID\_Array = newArray("WB\_Red.jpg","WB\_Green.jpg","WB\_Blue.jpg");

WB\_Col\_Array = newArray(WB\_Red\_Max, WB\_Green\_Max, WB\_Blue\_Max);

for(iMax = 0; iMax < WBcolourArray.length; iMax++)

{

// Calculate min and max of each colour channel in the white balance image

imageID = WBcolourArray[iMax];

maximageID = WB\_ID\_Array[iMax];

setcolorVal = WB\_Col\_Array[iMax];

selectWindow(imageID);

run("Select None");

run("Duplicate...", "title=MAX.JPG");

setColor(setcolorVal);

run("Select All");

fill(); // Fill a image window wih the WB\_Red\_Max colour

run("Select None");

imageCalculator("Subtract create", "MAX.JPG", imageID); // Create lighting uniformity correction image for the red channel

selectWindow("Result of MAX.JPG");

rename(maximageID);

run("Gaussian Blur...", "sigma=5"); // Blur to smooth correction image

run("Size...", "width=1780 height=1780 constrain interpolation=Bilinear"); // Re-size to original

saveAs("Jpeg", dir1 + maximageID); // Save correction image for future use

//Crop the uniformity image to have the same dimensions as the X-rite card

makeRectangle(XLeft1, XTop1, XWidth1, XHeight1);

run("Crop"); // Crop to match X-rite chart

run("Arbitrarily...", "angle="+Degree2+" grid=1 interpolation=Bilinear"); // Re-orient to match X-rite chart

makeRectangle(CropLeft, CropTop, CropWidth, CropHeight);

run("Crop"); // Crop to match X-rite chart

selectWindow(imageID);

close();

selectWindow("MAX.JPG");

close();

}

// ===== CORRECT THE ASSYMETRY OF LIGHTING (NON-UNIFORM) WITHIN EACH COLOUR CHANNEL OF THE IMAGE ===== //

imageCalculator("Add create", "Xrite.JPG (blue)","WB\_Blue.jpg"); // Add lighting uniformity correction image to chart image

rename("Normalized\_Blue");

resetThreshold;

ixstart = (Width - 30)/ 2 - 1;

ixinc = (CropWidth - Width) / 13;

icstart = ((Height - 30) / 2) - 5;

icinc = ((CropHeight - Height) / 9) - 15;

CellNum = 0;

CellList = newArray(141); // Don't use CellList[0] to make it easier to match up the cells.

for(ic=icstart; ic<CropHeight; ic += icinc)

{

for(ix=ixstart; ix< CropWidth; ix += ixinc, ic++)

{

setKeyDown("shift"); makeRectangle(ix,ic,30,30);

// Check each square for saturation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CellNum = 1 + CellNum;

CellList[CellNum] = 1;

b = getPixel(ix+15,ic+15);

if(b > 254) { CellList[CellNum] = 0; }

} // Put rectangles in the center of each square in X-rite chart

}

run("Create Mask"); // Make a mask of the center of all squares

selectWindow("Normalized\_Blue");

run("Clear Results");

run("Measure");

Xrite\_Min= getResult("Min", 0); // Get minimum gray valve of squares

Xrite\_Max=getResult("Max", 0); // Get maximum gray value of squares

run("Make Inverse");

setColor(Xrite\_Min - 2);

fill(); // Clear region outside of the mask

resetMinAndMax; // Resets values for the display

run("Make Inverse");

setThreshold(Xrite\_Min, Xrite\_Max);

run("Set Measurements...", "area centroid median redirect=None decimal=3");

run("Analyze Particles...", "size=100-Infinity circularity=0.00-1.00 show=Nothing display clear");

selectWindow("Results");

saveAs("Measurements", dir1+"Xrite\_Chart\_NormalizedBlueValues.txt"); // Save median gray values of all squares

selectWindow("Xrite.JPG (blue)");

close();

selectWindow("WB\_Blue.jpg");

close();

// Process Green X-rite chart image

imageCalculator("Add create", "Xrite.JPG (green)","WB\_Green.jpg"); // Add lighting uniformity correction image to chart image

rename("Normalized\_Green");

selectWindow("Normalized\_Green");

resetThreshold;

CellNum = 0;

for(ic=icstart; ic<CropHeight; ic += icinc)

{ for(ix=ixstart; ix< CropWidth; ix += ixinc, ic++)

{

// Check each square for saturation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CellNum = 1 + CellNum;

g = getPixel(ix+15,ic+15);

if(g > 254) { CellList[CellNum] = 0; } // print("g: "+CellNum); }

} //

}

selectWindow("Normalized\_Green");

resetThreshold;

run("Restore Selection");

run("Clear Results");

run("Set Measurements...", "area mean standard min centroid bounding redirect=None decimal=3");

run("Measure");

Xrite\_Min= getResult("Min", 0); // Get minimum gray value of squares

Xrite\_Max=getResult("Max", 0); // Get maximum gray value of squares

run("Make Inverse");

setColor(Xrite\_Min - 2);

fill(); // Clear region outside of the mask

resetMinAndMax; // Resets the values for the display

run("Make Inverse");

setThreshold(Xrite\_Min, Xrite\_Max);

run("Set Measurements...", "area centroid median redirect=None decimal=3");

run("Analyze Particles...", "size=100-Infinity circularity=0.00-1.00 show=Nothing display clear");

selectWindow("Results");

saveAs("Measurements", dir1+"Xrite\_Chart\_NormalizedGreenValues.txt"); // Save median gray values of all squares

selectWindow("Xrite.JPG (green)");

close();

selectWindow("WB\_Green.jpg");

close();

// Process Red X-rite chart image

imageCalculator("Add create", "Xrite.JPG (red)","WB\_Red.jpg"); // Add lighting uniformity correction image to chart image

rename("Normalized\_Red");

selectWindow("Normalized\_Red");

resetThreshold;

CellNum = 0;

for(ic=icstart; ic<CropHeight; ic += icinc)

{ for(ix=ixstart; ix< CropWidth; ix += ixinc, ic++)

{

// Check each square for saturation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CellNum = 1 + CellNum;

r = getPixel(ix+15,ic+15);

if(r > 254) { CellList[CellNum] = 0; } // print("r: "+CellNum); }

} // Put rectangles in the center of each square in X-rite chart

}

selectWindow("Normalized\_Red");

resetThreshold;

run("Restore Selection");

run("Clear Results");

run("Set Measurements...", "area mean standard min centroid bounding redirect=None decimal=3");

run("Measure");

Xrite\_Min= getResult("Min", 0); // Get minimum gray value of squares

Xrite\_Max=getResult("Max", 0); // Get maximum gray value of squares

run("Make Inverse");

setColor(Xrite\_Min - 2);

fill(); // Clear region outside of the mask

resetMinAndMax; // Resets the value for the display

run("Make Inverse");

setThreshold(Xrite\_Min, Xrite\_Max);

run("Set Measurements...", "area centroid median redirect=None decimal=3");

run("Analyze Particles...", "size=100-Infinity circularity=0.00-1.00 show=Nothing display clear");

selectWindow("Results");

saveAs("Measurements", dir1+"Xrite\_Chart\_NormalizedRedValues.txt"); // Save median gray values of all squares

selectWindow("Xrite.JPG (red)");

close();

selectWindow("WB\_Red.jpg");

close();

// Put normalized red, green & blue X-rite chart images back together to create the corrected color image.

run("Conversions...", " "); // This command forces ImageJ to convert 32-bit images to 8-bit images without UNWANTED scaling;

selectWindow("Normalized\_Red");

resetThreshold;

run("Select None");

selectWindow("Normalized\_Green");

resetThreshold;

run("Select None");

selectWindow("Normalized\_Blue");

resetThreshold;

run("Select None");

run("Merge Channels...", "red=Normalized\_Red green=Normalized\_Green blue=Normalized\_Blue gray=\*None\*");

// Now run the CORRECTED ImageJ Color Transformer plug-in

// The original came from http://rsbweb.nih.gov/ij/plugins/color-transforms.html

// The corrected version must be put into the plugins folder (or a subfolder) in order for the macro to run.

// This plugin converts an RGB color image into a color space for color analysis. The XYZ, Yxy, YUV, YIQ, Luv, Lab, AC1C2,

// I1I2I3, Yuv, YQ1Q2, HSI, HSV, HSL, LCHLuv, LSHLuv and LCHLab color spaces are supported. The lab option is used to

// convert into the CIE L\*, a\*, b\* color space.

selectWindow("RGB");

//run("Color Transformer Corrected", "colour=Lab");

run("Color Transformer 2", "from=sRGB to=Lab convert display");

selectWindow("Mask");

run("Create Selection");

selectWindow("RGB (Lab)"); // Peter Add

rename("Lab"); // Peter Add

selectWindow("Lab");

run("Restore Selection");

run("Clear Results");

run("Set Measurements...", "mean min centroid redirect=None decimal=3");

run("Measure");

Xrite\_Min= getResult("Min", 0);

Xrite\_Max=getResult("Max", 0);

setThreshold(Xrite\_Min, Xrite\_Max);

run("Set Measurements...", "area centroid median redirect=None decimal=3");

run("Analyze Particles...", "size=100-Infinity circularity=0.00-1.00 show=Nothing display clear slice");

saveAs("Measurements", dir1+"Xrite\_Chart\_NormalizedLSTARValues.txt"); // Save L\* values for calibration

resetThreshold;

selectWindow("Lab");

run("Next Slice [>]");

selectWindow("Mask");

run("Create Selection");

selectWindow("Lab");

run("Restore Selection");

run("Clear Results");

run("Set Measurements...", "mean min centroid redirect=None decimal=3");

run("Measure");

Xrite\_Min= getResult("Min", 0);

Xrite\_Max=getResult("Max", 0);

run("Make Inverse");

setColor(Xrite\_Min - 2);

fill();

resetMinAndMax;

run("Make Inverse");

setThreshold(Xrite\_Min, Xrite\_Max);

run("Set Measurements...", "area centroid median redirect=None decimal=3");

run("Analyze Particles...", "size=100-Infinity circularity=0.00-1.00 show=Nothing display clear slice");

saveAs("Measurements", dir1+"Xrite\_Chart\_NormalizedASTARValues.txt"); // Save a\* values for calibration

resetThreshold;

selectWindow("Lab");

run("Next Slice [>]");

selectWindow("Mask");

run("Create Selection");

selectWindow("Lab");

run("Restore Selection");

run("Clear Results");

run("Set Measurements...", "mean min centroid redirect=None decimal=3");

run("Measure");

Xrite\_Min= getResult("Min", 0);

Xrite\_Max=getResult("Max", 0);

run("Make Inverse");

setColor(Xrite\_Min - 2);

fill();

resetMinAndMax;

run("Make Inverse");

setThreshold(Xrite\_Min, Xrite\_Max);

run("Set Measurements...", "area centroid median redirect=None decimal=3");

run("Analyze Particles...", "size=100-Infinity circularity=0.00-1.00 show=Nothing display clear slice");

saveAs("Measurements", dir1+"Xrite\_Chart\_NormalizedBSTARValues.txt"); // Save b\* values for calibration

resetThreshold;

// \*\*\*\*\*\*\*\*\* Repeat the color transformer step to get the L\* C\* H\* images

selectWindow("RGB");

//run("Color Transformer Corrected", "colour=LCHLab"); // Corrected Color Transformer for L\* C\* H\*

run("Color Transformer 2", "from=sRGB to=LCHLab convert display");

selectWindow("RGB (LCHLab)"); // Peter Add

rename("LCHLab"); // Peter Add

selectWindow("LCHLab");

run("Next Slice [>]"); // Skip L\* image and go to C\* image \*\*\*\*\*\*NOTE Original Color Transformer is mislabeled \*\*\*

selectWindow("Mask");

run("Create Selection");

selectWindow("LCHLab");

run("Restore Selection");

run("Clear Results");

run("Set Measurements...", "mean min centroid redirect=None decimal=3");

run("Measure");

Xrite\_Min= getResult("Min", 0);

Xrite\_Max=getResult("Max", 0);

run("Make Inverse");

setColor(Xrite\_Min - 2);

fill();

resetMinAndMax;

run("Make Inverse");

setThreshold(Xrite\_Min, Xrite\_Max);

run("Set Measurements...", "area centroid median redirect=None decimal=3");

run("Analyze Particles...", "size=100-Infinity circularity=0.00-1.00 show=Nothing display clear slice");

saveAs("Measurements", dir1+"Xrite\_Chart\_NormalizedCSTARValues.txt"); // Save H\* values for calibration

resetThreshold;

selectWindow("LCHLab");

run("Next Slice [>]"); // Go to H\* image \*\*\*\*\*NOTE Original Color Transformer is mislabeled

selectWindow("Mask");

run("Create Selection");

selectWindow("LCHLab");

run("Restore Selection");

run("Clear Results");

run("Set Measurements...", "mean min centroid redirect=None decimal=3");

run("Measure");

Xrite\_Min= getResult("Min", 0);

Xrite\_Max=getResult("Max", 0);

run("Make Inverse");

setColor(Xrite\_Min - 2);

fill();

resetMinAndMax;

run("Make Inverse");

setThreshold(Xrite\_Min, Xrite\_Max);

run("Set Measurements...", "area centroid median redirect=None decimal=3");

run("Analyze Particles...", "size=100-Infinity circularity=0.00-1.00 show=Nothing display clear slice");

saveAs("Measurements", dir1+"Xrite\_Chart\_NormalizedHSTARValues.txt"); // Save C\* values for calibration

resetThreshold;

selectWindow("Mask");

close();

selectWindow("Lab");

close();

selectWindow("LCHLab");

close();

selectWindow("RGB");

close();

// ===== READ IN THE DATA FROM THE MINOLTA CALIBRATION FILE AND USE THIS TO GENERATE SOME REGRESSIONS ===== //

// \*\*\*\* READ IN THE MINOLTA XYZ and LAB Color Calibration Data \*\*\*\* ;

CalPath = dir2 + "Minolta\_Lab\_Cal\_Values\_Imagej\_order.txt";

MyStr = File.openAsString(CalPath);

CalData=split(MyStr,"\n\t");

End = lengthOf(CalData);

if(End == 1) setKeyDown("Esc");

// \*\*\*\* Read In the Normalized L-star Color Values Extracted above \*\*\*\*;

MyPath = dir1+"Xrite\_Chart\_NormalizedLSTARValues.txt";

MyStr = File.openAsString(MyPath);

LStar=split(MyStr,"\n\t");

End = lengthOf(LStar);

if(End == 1) setKeyDown("Esc");

// \*\*\*\* Read In the Normalized a-star Color Values Extracted above \*\*\*\*;

MyPath = dir1+"Xrite\_Chart\_NormalizedASTARValues.txt";

MyStr = File.openAsString(MyPath);

aStar=split(MyStr,"\n\t");

End = lengthOf(aStar);

if(End == 1) setKeyDown("Esc");

// \*\*\*\* Read In the Normalized b-star Color Values Extracted above \*\*\*\*;

MyPath = dir1+"Xrite\_Chart\_NormalizedBSTARValues.txt";

MyStr = File.openAsString(MyPath);

bStar=split(MyStr,"\n\t");

End = lengthOf(bStar);

if(End == 1) setKeyDown("Esc");

// \*\*\*\* Read In the Normalized H-star Color Values Extracted above \*\*\*\*;

MyPath = dir1+"Xrite\_Chart\_NormalizedHSTARValues.txt";

MyStr = File.openAsString(MyPath);

HStar=split(MyStr,"\n\t");

End = lengthOf(bStar);

if(End == 1) setKeyDown("Esc");

// \*\*\*\* Read In the Normalized C-star Color Values Extracted above \*\*\*\*;

MyPath = dir1+"Xrite\_Chart\_NormalizedCSTARValues.txt";

MyStr = File.openAsString(MyPath);

CStar=split(MyStr,"\n\t");

End = lengthOf(bStar);

if(End == 1) setKeyDown("Esc");

// Initialize variables for linear regression

SumXYL = 0.0;

SumXXL = 0.0;

SumXL = 0.0;

SumYL = 0.0;

SumXYa = 0.0;

SumXXa = 0.0;

SumXa = 0.0;

SumYa = 0.0;

SumXYb = 0.0;

SumXXb = 0.0;

SumXb = 0.0;

SumYb = 0.0;

SumXYH = 0.0;

SumXXH = 0.0;

SumXH = 0.0;

SumYH = 0.0;

SumXYC = 0.0;

SumXXC = 0.0;

SumXC = 0.0;

SumYC = 0.0;

CellCount=0;

HueCount=0;

CellNum=0;

//Calculate regression values

for (ic=19, ix=9; ix<End; ic+=10, ix+=5) // loop through data, calculating regression values

{

CellNum = 1 + CellNum;

yL = 0.0 + CalData[ic-2];

xL = 0.0 + LStar[ix];

ya = 0.0 + CalData[ic-1];

xa = 0.0 + aStar[ix];

yb = 0.0 + CalData[ic];

xb = 0.0 + bStar[ix];

yH = 0.0 + atan2(yb, ya);

if(yH < 0) yH = 2\*PI + yH;

yH = 180 \* yH / PI;

xH = 0.0 + HStar[ix];

yC = 0.0 + sqrt(yb\*yb + ya\*ya);

xC = 0.0 + CStar[ix];

// Values of L\* above 517 are likely saturated and unreliable

// Any red, green, or blue values above 254 are also eliminated

if ( (xL < 518) && CellList[CellNum])

{

CellCount = 1 + CellCount;

SumXYL = SumXYL + xL \* yL;

SumXL = SumXL + xL;

SumYL = SumYL + yL;

SumXXL = SumXXL + xL \* xL;

SumXYa = SumXYa + xa \* ya;

SumXa = SumXa + xa;

SumYa = SumYa + ya;

SumXXa = SumXXa + xa \* xa;

SumXYb = SumXYb + xb \* yb;

SumXb = SumXb + xb;

SumYb = SumYb + yb;

SumXXb = SumXXb + xb \* xb;

// Cells with C\* values below 45 must be omitted from the H\* calibration

if(xC > 44)

{

HueCount = 1 + HueCount;

SumXYH = SumXYH + xH \* yH;

SumXH = SumXH + xH;

SumYH = SumYH + yH;

SumXXH = SumXXH + xH \* xH;

}

SumXYC = SumXYC + xC \* yC;

SumXC = SumXC + xC;

SumYC = SumYC + yC;

SumXXC = SumXXC + xC \* xC;

}

}

SlopeL = (CellCount \* SumXYL - SumXL \* SumYL)/(CellCount \* SumXXL - SumXL \* SumXL);

InterceptL = (SumYL - SlopeL \* SumXL)/CellCount;

print("Intercept Slope");

print(InterceptL, SlopeL); // Display regression parameters for L\*

Slopea = (CellCount \* SumXYa - SumXa \* SumYa)/(CellCount \* SumXXa - SumXa \* SumXa);

Intercepta = (SumYa - Slopea \* SumXa)/CellCount;

print(Intercepta, Slopea); // Display regression parameters for a\*

Slopeb = (CellCount \* SumXYb - SumXb \* SumYb)/(CellCount \* SumXXb - SumXb \* SumXb);

Interceptb = (SumYb - Slopeb \* SumXb)/CellCount;

print(Interceptb, Slopeb); // Display regression parameters for b\*

SlopeC = (CellCount \* SumXYC - SumXC \* SumYC)/(CellCount \* SumXXC - SumXC \* SumXC);

InterceptC = (SumYC - SlopeC \* SumXC)/CellCount;

print(InterceptC, SlopeC); // Display regression parameters for C\*

SlopeH = (HueCount \* SumXYH - SumXH \* SumYH)/(HueCount \* SumXXH - SumXH \* SumXH);

InterceptH = (SumYH - SlopeH \* SumXH)/HueCount;

print(InterceptH, SlopeH); // Display regression parameters for H\*

print("BackgroundColor:", BackgroundColor); // Print the color of the background

print("SquarePixelWidth:", SquareWidth, mmWidth); // Print pixel square size for scale based on width

print("SquarePixelHeight:", SquareHeight, mmHeight); // Print pixel square size for scale based on height

print("XRiteScaleLength:", ScalePixelWidth, mmScale); // Print pixel length size based upon scale at bottom of chart

print("ImageScales:", MyXSizeVal, MyYSizeVal); // Print the image size values for size standardization

print("SizeFlag:", SizeFlag); // Print the Size Standardization Flag

// Create Label Mask Image \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (UseLabel > 0) {

open(dir1 + "Label.JPG");

run("Median...", "radius=3"); // Smooth to reduce noise

if( SizeFlag ) run("Size...", "width=1780 height=1780 average interpolation=Bilinear"); // Standardize Image Size

run("Duplicate...", "title=Label\_Temp.JPG"); // Make a temporary copy to use in finding chart placement in image

selectWindow("Label.JPG");

run("Split Channels");

selectWindow("Label.JPG (green)");

close();

selectWindow("Label.JPG (red)");

close();

selectWindow("Label.JPG (blue)");

run("Duplicate...", "title=Label\_Temp");

run("Select All");

setColor(0);

run("Fill");

run("Select None");

selectWindow("Label.JPG (blue)");

setAutoThreshold("Minimum dark");

run("Set Measurements...", "area centroid bounding redirect=None decimal=3");

run("Analyze Particles...", "size=10000-Infinity circularity=0.00-1.00 show=Nothing display clear");

numObs = nResults;

selectWindow("Label\_Temp");

setColor(255);

for (ix=0; ix<numObs; ix+=1)

{

LX=getResult("BX", ix); //X coordinate of top left

LY=getResult("BY", ix); //Y coordinate of top left

LWidth=getResult("Width", ix); //Width

LHeight=getResult("Height", ix); //Height

if (UseLabel == 2) {

LX = LX - LWidth\*0.25;

LY = LY - LHeight\*0.25;

makeRectangle(LX, LY, LWidth\*1.5, LHeight\*1.5);

}

else makeRectangle(0, 0, LX + LWidth\*1.5, 1780); // removed LHeight\*2); to switch to full height label region

fill();

run("Select None");

}

saveAs("Jpeg", dir1 + "Label\_Mask.JPG");

rename("Label\_Temp");

selectWindow("Label.JPG (blue)");

close();

if (QRCode == 1) UseLabel = 1;

else UseLabel = 2;

run("Clear Results");

// Open White Balance images for illumination uniformity correction

open(dir1 + "WB\_Blue.JPG");

run("Set... ", "zoom=10");

selectWindow("Label\_Temp.JPG");

run("Split Channels");

selectWindow("Label\_Temp.JPG (blue)");

imageCalculator("Add create", "Label\_Temp.JPG (blue)","WB\_Blue.JPG"); //illumination offset adjustment.

rename("Normalized\_Blue");

selectWindow("Label\_Temp.JPG (blue)");

close();

selectWindow("WB\_Blue.JPG");

close();

selectWindow("Label\_Temp");

run("Create Selection");

selectWindow("Normalized\_Blue");

run("Restore Selection");

run("Make Inverse");

setColor(255);

fill(); // Clear ares in produce image corresponding to the label

makeRectangle(600, 200, 2200, 1900);

run("Make Inverse");

setColor(255);

fill(); // Clear ares in produce image corresponding to the label

run("Select None");

setThreshold(0, 254);

run("Create Selection");

run("Set Measurements...", "area mean standard min median redirect=None decimal=3");

run("Clear Results");

run("Measure");

close();

BlueMin=getResult("Median", 0) + 3\*round(getResult("StdDev") + 0.5);

open(dir1 + "WB\_Green.JPG");

run("Set... ", "zoom=10");

selectWindow("Label\_Temp.JPG (green)");

imageCalculator("Add create", "Label\_Temp.JPG (green)","WB\_Green.JPG"); //illumination offset adjustment.

rename("Normalized\_Green");

selectWindow("Label\_Temp.JPG (green)");

close();

selectWindow("WB\_Green.JPG");

close();

selectWindow("Label\_Temp");

run("Create Selection");

selectWindow("Normalized\_Green");

run("Restore Selection");

run("Make Inverse");

setColor(255);

fill(); // Clear ares in produce image corresponding to the label

makeRectangle(600, 200, 2200, 1900);

run("Make Inverse");

setColor(255);

fill(); // Clear ares in produce image corresponding to the label

run("Select None");

setThreshold(0, 254);

run("Create Selection");

run("Set Measurements...", "area mean standard min median redirect=None decimal=3");

run("Clear Results");

run("Measure");

close();

GreenMin=getResult("Median", 0) + 3\*round(getResult("StdDev") + 0.5);

open(dir1 + "WB\_Red.JPG");

run("Set... ", "zoom=10");

selectWindow("Label\_Temp.JPG (red)");

imageCalculator("Add create", "Label\_Temp.JPG (red)","WB\_Red.JPG"); //illumination offset adjustment.

rename("Normalized\_Red");

selectWindow("Label\_Temp.JPG (red)");

close();

selectWindow("WB\_Red.JPG");

close();

selectWindow("Label\_Temp");

run("Create Selection");

selectWindow("Normalized\_Red");

run("Restore Selection");

run("Make Inverse");

setColor(255);

fill(); // Clear ares in produce image corresponding to the label

makeRectangle(600, 200, 2200, 1900);

run("Make Inverse");

setColor(255);

fill(); // Clear ares in produce image corresponding to the label

run("Select None");

setThreshold(0, 254);

run("Create Selection");

run("Set Measurements...", "area mean standard min median redirect=None decimal=3");

run("Clear Results");

run("Measure");

close();

RedMin=getResult("Median", 0) + 3\*round(getResult("StdDev") + 0.5);

selectWindow("Label\_Temp");

close();

}

else {

RedMin = 25;

GreenMin = 25;

BlueMin = 25;

}

print("LabelFlag:", UseLabel); // Print the Label Flag

print("Red\_Min:", RedMin); // Print the estimated threshold value for produce

print("Green\_Min:", GreenMin); // Print the estimated threshold value for produce

print("Blue\_Min:", BlueMin); // Print the estimated threshold value for produce

// End of Label Mask section \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

selectWindow("Log");

saveAs("Text", dir1+"Log.txt"); // Save L\*, a\*, b\* calibration parameters for use with produce images

selectWindow("Log");

run("Close");

CellsOmitted = 140 - CellCount;

if ( CellsOmitted > 0) {

Dialog.create("Saturated Cells Warning");

Dialog.addMessage("The calibration was completed successfully.\n \n"

+"However, "+CellsOmitted+" XRite cells were omitted due to saturation.\n \n"

+"To avoid this in the future you can try moving\n"

+"the lamps slightly away from the photographic\n"

+"tent, or decreasing the shutter speed.");

Dialog.show();

}

print("MacBeth Calibration script has completed");

// END OF IMAGEJ COLOR CALIBRATION MACRO