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

/\* Major revision 01 April 2019 by Peter McAtee (peter.mcatee@plantandfood.co.nz)

\* Original macro developed 02 April 2018 by Irwin Donis-Gonzalez

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

/\* ImageJ macro revised for high throughput phenotyping of walnut kernels for Neale Lab at UC Davis.

\* It is possible to run this macro in "visible" mode where the images are displayed while being processed or in "invisible"

\* mode where the images are not displayed. The macro will run more quickly if the images are not displayed. By default \

\* the macro is set to display all images. Change the word "false" to "true" in later lines commands to switch modes.

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

\*/

/\* MACRO OBJECTIVE------------------------------------------------------------------------------

\* The function of this macro is to convert the sRGB values in the digital images of walnut into CIE L\*, a\*, b\* color values.

\*This macro requires that the Walnut\_Macbeth\_Calibrator\_tool\_V1.1 to be run prior for

\* the needed illumination uniformity correction images and the color calibration constants.

\* This version includes automatic centimeter scaling based upon XRite chart squares

\* This version includes label masking.

\* This version includes summary output and is setup for easier pivot table creation.

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

\* NOTE THIS REQUIRES THE CORRECTED Color Transformer plugin.

\*/

//Version 2.0

requires("1.52k");

// Set 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");

//Choose the folder that contains the calibration images. Must be the calibration images for the particular set of sample images processing

print("Please select the directory that contains the X-Rite chart, White Balance, & Label Images");

dir0 = getDirectory("Please select the directory that contains the X-Rite chart, White Balance, & Label Images");

//dir0 = "C:\\Users\\hrapym\\Desktop\\walnut\\Calibration\_12\_13\_18\\";

// Ask user to identify the folder containing the WALNUT images

print("Please select the directory that contains WALNUT Images");

dir1 = getDirectory("Please select the directory that contains WALNUT Images");

//dir1 = "C:\\Users\\hrapym\\Desktop\\walnut\\test\_walnut\\";

// Ask user to identify the folder containing the XRITE chart and White Balance images and the file of calibration constants

print("Please select a directory for outputs");

dir2 = getDirectory("Please select a directory for outputs");

//Creates a folder titled "Labeled Images" within the directory that will store the binary labeled images

splitDir = dir2 + "Labeled Images\\";

print(splitDir);

File.makeDirectory(splitDir);

//Create or choose a file to which the program will write the results. This file will override existing files; must choose a unique filename.

print("Creating a CSV file to save colormetric analysis to ... ");

//Get Time and Date

getDateAndTime(year, month, dayOfWeek, dayOfMonth, hour, minute, second, msec);

//Set the time

if (hour<10) {hour = "0"+hour;}

if (minute<10) {minute = "0"+minute;}

if (second<10) {second = "0"+second;}

print("Walnut colour quantification pipline started at- " +hour+":"+minute+ ":"+second);

//Set the date

month=month+1;

if (month<10) {month = "0"+month;}

if (dayOfMonth<10) {dayOfMonth = "0"+dayOfMonth;}

month=toString(month); dayOfMonth=toString(dayOfMonth, 0); year=toString(year);

myDATE = month+"."+dayOfMonth+"."+year;

myDATEOUT ="\_"+month+dayOfMonth+year+"\_";

//Define a name for output CSV file

myOutPutFile = "Walnut\_"+myDATEOUT+".csv";

//Define the header values for the physiological measurements

myPhys\_Title = "Image,Date,QRCode,Item#, Alphanumeric\_Coordinate,Walnut\_Detected, Area(cm2),Perimeter(cm),MajorDia(cm),MinorDia(cm),EllipseAngle(deg), X(cm),Y(cm),BoxWidth(cm),BoxHeight(cm)";

//Vector of headers for each colour dimension value

myL\_Value\_Title = ",L\_Mean, L\_Median, L\_Mode, L\_StdDev, L\_Min, L\_Max";

myA\_Value\_Title = ",a\_Mean, a\_Median, a\_Mode, a\_StdDev, a\_Min, a\_Max";

myB\_Value\_Title = ",b\_Mean, b\_Median, b\_Mode, b\_StdDev, b\_Min, b\_Max";

//Build master header vector

myTitle = myPhys\_Title+myL\_Value\_Title+myA\_Value\_Title+myB\_Value\_Title;

//Select name for output CSV file

//IF STATEMENT to write a file header if a file exists. Otherwise append to old file.

CSV\_TEST = File.exists(dir1+myOutPutFile);

if(CSV\_TEST == 0){

File.append(myTitle, dir1+myOutPutFile);

} else {

continue;

}

//Obtain some user inputs

Dialog.create("User input required");

Dialog.addNumber("Labeling Orienation? 1 = Column-wise; 2 = Row-wise", 1); //The user chooses the orientation of the labeling. Column-wise is the default and will label a column at a time. Row-wise will go through a row at a time.

Dialog.addNumber("Number of Rows?", 10); //The number of rows containing walnuts. If only x rows are filled, but they are adjacent, keep as 10.

Dialog.addNumber("Number of Columns?", 10); //The number of rows containing walnuts. If only y columns are filled, but they are adjacent, keep as 10.

Dialog.show();

label\_orientation = Dialog.getNumber();

rows = Dialog.getNumber();;

columns = Dialog.getNumber();;;

/\* Read In the Normalization & Calibration Values \*/

MyPath = dir0+"Log.txt";

MyStr = File.openAsString(MyPath);

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

End = lengthOf(Values);

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

/\* Programming note. Must use the = 0.0 + format to get ImageJ to automatically convert the string values in Values[i]

\* to numerical values.

\*/

//COLOR CALIBRATION VALUES------------------------------------------------

Lstar\_Intercept = 0.0 + Values[2];

Lstar\_Slope = 0.0 + Values[3];

Astar\_Intercept = 0.0 + Values[4];

Astar\_Slope = 0.0 + Values[5];

Bstar\_Intercept = 0.0 + Values[6];

Bstar\_Slope = 0.0 + Values[7];

Cstar\_Intercept = 0.0 + Values[8];

Cstar\_Slope = 0.0 + Values[9];

Hstar\_Intercept = 0.0 + Values[10];

Hstar\_Slope = 0.0 + Values[11];

if(End > 12) BackgroundColor = Values[13];

if(End > 15) { SquareWidth = 0.0 + Values[15]; SquareWidthMM = 0.0 + Values[16]; }

if(End > 18) { SquareHeight = 0.0 + Values[18]; SquareHeightMM = 0.0 + Values[19]; }

if(End > 21) { ScalePixels = 0.0 + Values[21]; ScaleLenghtMM = 0.0 + Values[22]; }

if(End > 24) { ScalePixelWidth = 0.0 + Values[24]; ScalePixelHeight = 0.0 + Values[25]; }

if(End > 26) { SizeFlag = 0.0 + Values[27]; }

if(End > 28) { LabelFlag = 0.0 + Values[29]; }

if(End > 30) { RedMin = 0.0 + Values[31]; }

if(End > 32) { GreenMin = 0.0 + Values[33]; }

if(End > 34) { BlueMin = 0.0 + Values[35]; }

MeanSquarePixels = (SquareWidth + SquareHeight)/2;

MeanSquareCM = (SquareWidthMM + SquareHeightMM)/20;

PixelsPerCm = MeanSquarePixels/MeanSquareCM;

WalnutLabel = 0;

// Ask user to choose data output

//The output is for the individual walnuts only. We can implement for the entire image later if needed.

//DataOutput = 0;

DataOutput = 2;

//DataOutput = getNumber("Choose result output (0 for results only by walnut, 1 for results only by image, 2 for both ): ", DataOutput);

// It is possible to run this macro in "visible" mode where the images are displayed while being processed or in "invisible"

// mode where the images are not displayed. The macro will run more quickly if the images are not displayed. By default

// 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(true); // keep in visible mode (false)

// Label image is used to show ImageJ where a label may be placed in order to ignore it during analysis

if (LabelFlag > 0)

{

open(dir0 + "Label\_Mask.JPG");

run("Convert to Mask");

run("Duplicate...", "title=LabelSmall");

run("Convert to Mask");

selectWindow("Label\_Mask.JPG");

close();

}

//Creating the time stamps for the images.

MonthNames = newArray("Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec");

DayNames = newArray("Sun", "Mon","Tue","Wed","Thu","Fri","Sat");

getDateAndTime(year, month, dayOfWeek, dayOfMonth, hour, minute, second, msec);

TimeString ="Analysis Date: "+DayNames[dayOfWeek]+" ";

if (dayOfMonth<10) TimeString = TimeString+"0";

TimeString = TimeString+dayOfMonth+"-"+MonthNames[month]+"-"+year+", Analysis Time: ";

if (hour<10) TimeString = TimeString+"0";

TimeString = TimeString+hour+":";

if (minute<10) TimeString = TimeString+"0";

TimeString = TimeString+minute+":";

if (second<10) TimeString = TimeString+"0";

TimeString = TimeString+second;

//Printing to the log screen.

print("UC Davis - BAE - Computer Vision System - Walnut Color Image Analysis" );

print("Program started at: "+TimeString);

print("Directory Selected = " + dir1);

/\*---------------------------------------------------Main Function-------------------------------------

\* This is where the function will get called in a loop to label each row at a time. After, labels will be applied to original image.

\* Counter keeps track of which row working in. This is because IMAGE-J does not support 2D arrays easily.

\*/

//Counter variable that will be used to count the number of walnuts in the image.

count = 0;

list = getFileList(dir1); //Gets the first image from the sample directory

open(dir1+list[0]);

run("Set Scale...", "distance=280 known=6 unit=mm"); //Setting the scale for the batch

run("Labels...", "color=white font=36 show draw"); //Setting the label size. Large size needed for flattened images to be legible.

w = getWidth()/columns;

h = getHeight()/rows;

selectWindow(list[0]);

close();

LCHoutput = 0;

//Creates the dimensions of the rectangular selection that will be used during segmentation. It is done here so that user is only prompted once and not during every loop iteration.;

/\*Main loop that will go through each of the images

\* This loop will take one image at a time, collect the color values, then go through the next file in the sample folder until all of the images have been processed

\*/

for (i = 0; i<list.length; i++)

{

if(endsWith(list[i], ".jpg") || endsWith(list[i], ".JPG"))

{

showProgress(i+1, list.length);

print("Processing image "+list[i]);

imgName = list[i];

open(dir1+imgName);

rename(imgName);

imgNameBase = substring(list[i], 0, lengthOf(list[i])-3);

run("Median...", "radius=3");

//QR Code for future use. This will be a column in the results output.

if (LabelFlag == 2)

{

run("QR Decoder", "error=NONE\_FOUND");

selectWindow("QR Code");

WindowText=getInfo("window.contents");

mylines = split(WindowText, "\n");

QRresult = mylines[0];

run("Close");

} else

{

QRresult = " ";

}

LastDate=File.dateLastModified(dir1 + imgName);

//Sets the measurements that will be taken on the image and on the ROIs

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

//Funtion that will rotate the image if the image taken in the Color Vision System was slightly tilted.

//imageRotation();

// PREPARE IMAGE TO FACILITATE BETTER THRESHOLD SEGMENTATION //

selectWindow(imgName);

imgThresh = imgNameBase+"\_Threshold";

//Clears the results so that previously collected data is not a part of the results output.

run("Clear Results");

run("Select None");

run("Duplicate...", "title=Image-1.jpg");

run("Duplicate...", "title=Image-2.jpg");

selectWindow("Image-2.jpg");

run("Gaussian Blur...", "sigma=20"); // Smooth the noise out of Image-2.jpg

imageCalculator("Subtract create", "Image-1.jpg","Image-2.jpg");

selectWindow("Result of " +"Image-1.jpg");

rename("Subtraction.jpg");

imageCalculator("Add create", "Image-1.jpg","Subtraction.jpg");

selectWindow("Result of "+"Image-1.jpg");

rename(imgThresh);

selectWindow("Image-2.jpg");

close();

selectWindow("Image-1.jpg");

close();

selectWindow("Subtraction.jpg");

close();

// COLOUR THRESHOLD //

run("Clear Results");

selectWindow(imgThresh);

setOption("BlackBackground", true);

min=newArray(3);

max=newArray(3);

filter=newArray(3);

a=getTitle();

run("HSB Stack");

run("Convert Stack to Images");

selectWindow("Hue");

rename("0");

selectWindow("Saturation");

rename("1");

selectWindow("Brightness");

rename("2");

min[0]=155; // 0

max[0]=190; //255

filter[0]="stop"; //pass

min[1]=40;

max[1]=255;

filter[1]="pass";

min[2]=30; // 40

max[2]=255;

filter[2]="pass";

for (Q=0;Q<3;Q++)

{

selectWindow(""+Q);

setThreshold(min[Q], max[Q]);

run("Convert to Mask");

if (filter[Q]=="stop") run("Invert");

}

imageCalculator("AND create", "0","1");

imageCalculator("AND create", "Result of 0","2");

for (Q=0;Q<3;Q++)

{

selectWindow(""+Q);

close();

}

selectWindow("Result of 0");

close();

selectWindow("Result of Result of 0");

rename(a);

// Colour Thresholding END-------------

selectWindow(imgThresh);

run("Make Binary");

//run("Convert to Mask");

//run("Invert");

//run("Close-");

run("Fill Holes");

run("Clear Results");

//This is a way of combating ImageJ's inability to work with nested for loops on an image.

// In a loop, a labeling function is called that will label the images row-wise or column-wise based on the variable label\_orientation

counter = 0;

for (j = 0; j < 10; j++)

{

numberWalnuts = rowLabel(label\_orientation,j,w,h,counter);

counter = counter+1;

}

// PLACE THE ROI OUTLINES ON A COPY OF THE ORIGINAL IMAGE //

//The ROI's from the duplicated image are placed on the original image (with the background removed).

// This image is converted into a mask, flattened, and saved to the Labeled Images" folder. The labeled image is closed for organizational purposes.

selectWindow(imgThresh);

run("Select None");

run("From ROI Manager");

run("Convert to Mask");

LabelsOnImage();

selectWindow(imgThresh);

run("Create Selection");

numberitems = roiManager("count");

imgDup = imgNameBase+"\_Dup";

imgLabel = imgNameBase+"\_Label";

selectWindow(imgName);

run("Duplicate...", "title="+imgDup);

selectWindow(imgDup);

run("From ROI Manager");

run("Flatten");

rename(imgLabel);

selectWindow(imgLabel);

saveAs("Jpeg", splitDir+imgLabel);

close();

selectWindow(imgDup);

close();

//The original image is split into three channels, green, blue, red. These images will then be analyzed to obtain the LABCH Channels.

selectWindow(imgName);

run("Split Channels");

// MAKE COLOUR CORRECTION TO LIGHT UNIFORMITY //

// Open White Balance images for illumination uniformity correction

open(dir0 + "WB\_Red.jpg");

open(dir0 + "WB\_Green.jpg");

open(dir0 + "WB\_Blue.jpg");

ColArray = newArray("\_Red", "\_Green", "\_Blue");

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

redChannel= "";

greenChannel = "";

blueChannel = "";

redChannel = imgName+" (red)";

greenChannel = imgName+" (green)";

blueChannel = imgName+" (blue)";

IMG\_ColArray = newArray(redChannel, greenChannel, blueChannel);

for(iCol =0; iCol < IMG\_ColArray.length; iCol++)

{

Channel = "";

imgChannel = "";

wbChannel = "";

Channel = ColArray[iCol];

imgChannel = IMG\_ColArray[iCol];

wbChannel = WB\_ColArray[iCol];

imageCalculator("Add create", imgChannel, wbChannel);

rename("Normalized"+Channel);

selectWindow(imgThresh);

run("Create Selection");

selectWindow("Normalized"+Channel);

selectWindow(imgChannel);

close();

}

selectWindow("WB\_Red.jpg");

close();

selectWindow("WB\_Green.jpg");

close();

selectWindow("WB\_Blue.jpg");

close();

// CONVERT IMAGE TO Lab COLOUR SPACE //

// Convert into L\*, a\*, b\* image //

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

selectWindow("Normalized\_Red");

run("Select None");

selectWindow("Normalized\_Green");

run("Select None");

selectWindow("Normalized\_Blue");

run("Select None");

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

selectWindow("RGB");

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

selectWindow("RGB (Lab)");

rename("Lab");

// Apply color calibration to image containing the walnut samples //

selectWindow("Lab");

setSlice(1);

run("Multiply...", "value="+Lstar\_Slope);

run("Add...", "value="+Lstar\_Intercept);

selectWindow("Lab");

setSlice(2);

run("Multiply...", "value="+Astar\_Slope);

run("Add...", "value="+Astar\_Intercept);

selectWindow("Lab");

setSlice(3);

run("Multiply...", "value="+Bstar\_Slope);

run("Add...", "value="+Bstar\_Intercept);

if(LCHoutput == 1)

{

// Convert into L\*, C\*, H\* image //

selectWindow("RGB");

run("Color Transformer 2", "from=sRGB to=LCHLab convert display"); // Corrected Color Transformer for L\* C\* H\*

selectWindow("RGB (LCHLab)");

rename("LCHLab");

selectWindow("LCHLab");

// Skip L\* image and go to C\* image \*\*\*\*\*\*NOTE Original Color Transformer is mislabeled \*\*\* //

selectWindow("LCHLab");

setSlice(2);

run("Multiply...", "value="+Cstar\_Slope);

run("Add...", "value="+Cstar\_Intercept);

selectWindow("LCHLab");

setSlice(3);

run("Multiply...", "value="+Hstar\_Slope);

run("Add...", "value="+Hstar\_Intercept);

selectWindow("LCHLab");

run("Stack to Images");

}

selectWindow("RGB");

close();

selectWindow(imgThresh);

close();

selectWindow("Lab");

run("Stack to Images");

// Use the L\*, a\*, b\*, C\*, & H\* channels to extract colormetric values from walnut items //

itemArray = newArray();

for (item = 0; item < numberitems; item++)

{

itemArray = Array.concat(itemArray , item);

}

//Make array of colour windows depending on what output was selected

if(LCHoutput == 1)

{

windowNameArray = newArray("L","a","b","C","H");

}

else

{

windowNameArray = newArray("L","a","b");

}

// RECURCE THROUGH EACH WALNUT BY COLOR CHANNEL TO EXTRACT COLOMETRIC VALUES //

myResults = numberitems;

setColor(0);

setFont("SansSerif", 200, "bold");

setJustification("center");

n = numberitems; //Note the number of items as n to keep tidy

if (DataOutput != 1)

{

//Placing the AlphaNumeric System for labeling. This is repeated for all color channels.

for (n=0; n < myResults; n++)

{

if (n < 10)

{

alphanumeric = "A";

}

if (n >= 10 && n < 20)

{

alphanumeric = "B";

}

if (n >= 20 && n < 30)

{

alphanumeric = "C";

}

if (n >= 30 && n < 40)

{

alphanumeric = "D";

}

if (n >= 40 && n < 50)

{

alphanumeric = "E";

}

if (n >= 50 && n < 60)

{

alphanumeric = "F";

}

if (n >= 60 && n < 70)

{

alphanumeric = "G";

}

if (n >= 70 && n < 80)

{

alphanumeric = "H";

}

if (n >= 80 && n < 90)

{

alphanumeric = "I";

}

if (n >= 90)

{

alphanumeric = "J";

}

number = n+1;

alpha\_num = n%10+1;

coordinate = alphanumeric + alpha\_num; //This is the alphanumeric coordinate that will be recorded in the results.

number = n+1;

myBoolDection = "FALSE";

for(iLAB =0; iLAB < windowNameArray.length; iLAB++)

{

windowName = "";

windowName = windowNameArray[iLAB];

myProcessingIMG = "Process\_IMG";

selectWindow(windowName);

run("Duplicate...", "title="+myProcessingIMG);

run("Set Scale...", "distance=280 known=6 unit=mm"); //Setting the scale for the batch

selectWindow(myProcessingIMG);

run("Clear Results"); //Clear previous results so that only relevant results are recorded.

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

selectWindow(myProcessingIMG);

roiManager("Select", n);

List.clear();

List.setMeasurements();

ANGLE = List.get("Angle");

selectWindow(myProcessingIMG);

close();

// If there is variance in the sub-image greater than 0.0005 this probably is a walnut and colour values are obtained

if (ANGLE > 0)

{

myBoolDection = "TRUE";

MEAN = "";

MEDIAN = "";

MODE = "";

STDEV = "";

MIN = "";

MAX = "";

MEAN = toString(List.get("Mean"));

MEDIAN = toString(List.get("Median"));

MODE = toString(List.get("Mode"));

STDEV = toString(List.get("StdDev"));

MIN = toString(List.get("Min"));

MAX = toString(List.get("Max"));

if (iLAB == 0)

{

AREA = "";

PERIM = "";

MAJOR = "";

MINOR = "";

ANGLE = "";

XCO = "";

YCO = "";

WIDTH = "";

HEIGHT = "";

AREA = toString(List.get("Area"));

PERIM = toString(List.get("Perim."));

MAJOR = toString(List.get("Major"));

MINOR = toString(List.get("Minor"));

ANGLE = toString(List.get("Angle"));

XCO = toString(List.get("X"));

YCO = toString(List.get("Y"));

WIDTH = toString(List.get("Width"));

HEIGHT = toString(List.get("Height"));

//Build colour output vector for metadata and physiological measurements

myPhysOutput = "";

myPhysOutput = imgName+","+LastDate+","+QRresult+","+number+","+coordinate+","+myBoolDection+","

+AREA+","+PERIM+","+MAJOR+","+MINOR+","+ANGLE+","+XCO+","+YCO+","+WIDTH+","+HEIGHT;

// Build colour output vector for each walnut in the L colour dimension

myL\_DimOutput = "";

myL\_DimOutput = MEAN+","+MEDIAN+","+MODE+","+STDEV+","+MIN+","+MAX;

}

else if (iLAB == 1)

{

// Build colour output vector for each walnut in the a colour dimension

myA\_DimOutput = "";

myA\_DimOutput = MEAN+","+MEDIAN+","+MODE+","+STDEV+","+MIN+","+MAX;

}

else if (iLAB == 2)

{

// Build colour output vector for each walnut in the b colour dimension

myB\_DimOutput = "";

myB\_DimOutput = MEAN+","+MEDIAN+","+MODE+","+STDEV+","+MIN+","+MAX;

}

}

else

{

// IF the variance in the subimage is extremely small (<= 0.005) then the cell is empty. 0 values will be substituted

myBoolDection = "FALSE";

//Build NULL output vectors for NO walnut positions

myPhysOutput = list[i]+","+LastDate+","+QRresult+","+number+","+coordinate+","+myBoolDection+",0,0,0,0,0,0,0,0,0";

myL\_DimOutput = "0,0,0,0,0,0";

myA\_DimOutput = "0,0,0,0,0,0";

myB\_DimOutput = "0,0,0,0,0,0";

}

}

myOUTPUT\_line ="";

myOUTPUT\_line = myPhysOutput+","+myL\_DimOutput+","+myA\_DimOutput+","+myB\_DimOutput;

File.append(myOUTPUT\_line, dir1+myOutPutFile);

} // END FOR LOOP

}

for(iLAB =0; iLAB < windowNameArray.length; iLAB++)

{

windowNameClose = "";

windowNameClose = windowNameArray[iLAB];

selectWindow(windowNameClose);

close();

}

//selectWindow(imgThresh);

//close();

roiManager("Deselect"); //Deletes the ROI manager so it is cleared when processing the next image.

roiManager("Delete");

}

}

print ("Script has completed");

/\*-------------------------------------------------------------------Color Threshold-----------------------------------------------

/\*--------------------------------------------------------------Rotation Function------------------------------------------------------------------------\*/

function imageRotation()

{

rotation = getNumber("Is the image tilted? 1 = Yes; 2 = No", 2);

if (rotation == 1)

{

rotationAngle = getNumber("CW or CCW? 1 = CW; 2 = CCW", 1);

if (rotationAngle == 1)

{

run("Rotate... ", "angle=-1 grid=1 interpolation=Bilinear");

}

if (rotationAngle == 2)

{

run("Rotate... ", "angle=1 grid=1 interpolation=Bilinear");

}

}

}

/\*This function will tilt the image Clockwise or counter clockwise depending on the correction needed. It only works for slightly tilted images. \*/

/\*--------------------------------------------------------------Labeling Function------------------------------------------------------------------------\*/

function rowLabel(label\_orientation,j,w,h,counter)

{

for (i = 0; i < 10; i++)

{

if(label\_orientation == 1)

{

x\_1 = h\*counter; //Creates the X and Y positions for the rectangular selection

y\_1 = i\*(w);

}

if (label\_orientation == 2)

{

y\_1 = h\*counter;

x\_1 = i\*(w);

}

makeRectangle(x\_1, y\_1, h, w); //Creates a rectangular selection around a single space in the grid.

run("Analyze Particles...", "size=2000-Infinity circularity=0.05-1.00 redirect=None include add");

nROIs = roiManager("count");

// ==== Get the number of ROIs within the rectangular selection. If there are more ROIs than the number of walnuts, then the walnut pieces are combined. ==== //

numResult = nResults;

run("Clear Results");

if(numResult > 1)

{

Count = numResult;

particles = 1;

while (particles < Count)

{

a = nROIs - Count;

b = nROIs - Count+1;

roiManager("Deselect");

roiManager("Select", newArray(a,b));

roiManager("Combine");

roiManager("Update");

//roiManager("Measure");

roiManager("Deselect");

roiManager("Select", b);

roiManager("Delete");

particles = particles +1;

}

nROIs = roiManager("count");

} else

{

nROIs = roiManager("count");

// ==== If there is no ROI in a grid location, a rectangle is inserted as a placeholder. This enables empty boxes to be labeled ==== //

if (nROIs < 10\*counter +i+1)

{

makeRectangle(x\_1, y\_1, 7\*h/8, 7\*w/8);

setForegroundColor(255, 255, 255);

run("Fill", "slice");

run("Draw", "slice");

run("Analyze Particles...", "size=2000-Infinity circularity=0.05-1.00 include add");

nROIs = roiManager("count");

makeRectangle(x\_1, y\_1, 7\*h/8, 7\*w/8);

setForegroundColor(255,255, 255);

run("Fill", "slice");

run("Draw", "slice");

}

}

ROI\_number\_ID = nROIs-1;

roiManager("Select", ROI\_number\_ID);

roiManager("Rename", nROIs);

roiManager("Deselect");

//IJ.deleteRows(0, 1); // This line causes the function to display the results window

run("Clear Results");

}

return(nROIs);

}

/\*--------------------------------------------------------------Labels on Image----------------------------------------------------------------\*/

//Here, the labels have all been created on Image. Now, the user is asked if they want the labels to be added to the original image. If yes, the overlay is added to the original image. \*/

function LabelsOnImage()

{

roiManager("Show All without labels");

run("Restore Selection");

roiManager("Show All with labels");

roiManager("Associate", "true");

roiManager("Centered", "false");

roiManager("UseNames", "false");

}