COLOR BASED CLASSIFICATION OF CIRCULAR MARKERS FOR THE IDENTIFICATION OF EXPERIMENTAL UNITS

by

LAKSHMI NARJALA

B.E., University Of Visveswaraya College of Engineering, Bangalore, India, 2010

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing And Information Sciences
College Of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2013

Approved by:

Major Professor
Dr. Daniel Andresen

Abstract

The purpose of this project is to analyze the growth of plants under certain lighting conditions. In order to ensure ideal lighting for all plants under demanding conditions like lack of optimal light due to shadowing, side wall reflections, overlapping of plants, etc., pots are rotated manually in an irregular fashion. To keep track of the position of these plants from time to time, a marking system is used for each tray of 16 plants. These markers are unique for each tray High definition surveillance cameras placed above these plants capture the plant images periodically. These images undergo image processing. Image processing should be able to identify and recognize the plants from the identification markers that were placed within each tray and thereby draw the statistics about the growth of the plants. Hence the computing part of this project is all about extracting the identity of a plant through image processing.

Image processing involves object and color recognition. Fiji, an image processing tool, is used for object recognition and the Python image module called “Image” is used for color recognition. Object recognition accurately locates the position of these circular objects and measures their size and shape. Color recognition identifies the pixel values of these circular objects. Finally the code corresponding to three-element groups of these circular units is fetched and stored. This code gives the identity of the tray and, therefore, each plant.

The timestamp that is stored with each plant image along with the code fetched through image processing is used to track the location of a plant in the plant chamber through time.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I take this opportunity to extend my sincere thanks to **Dr. Daniel Andresen**, my major professor for his continuous guidance throughout my research project and graduate studies. He played a significant role in the successful completion of this project, and it has been an honor to work with him.

I would like to express my deepest gratitude to **Dr. Stephen Welch**, my committee member for giving me an opportunity to work on this project. His guidance, timely advice and suggestions have helped me improve my research skills and become a better programmer.

I would like to express my sincere gratitude to **Dr. Mitchell Neilsen**, my committee member for his guidance and support in the successful completion of my academics.

My special thanks are extended to **Dr. Christine Palmer** from the Department of Plant Biology, University of California, Davis who is in charge of the experiment of which this project is a supporting part, for all the information, help and images she supplied.

My heartfelt thanks to the faculty and staff of Department of Computer and Information Sciences for their unflinching support, and to all my friends for the wonderful memories that made my stay in Manhattan, Kansas beautiful and cherishing.

I would also like to express my thanks and appreciation to the National Science Foundation, which funded this work under grant #0923752 to the University of Wyoming, Dartmouth College, the University of California at Davis, and Kansas State University.

Dedication

I dedicate this project report to my parents, N.Vasudeva Bhatt and Sumangala Bhatt. Without your blessings and support I would not have been what I am today.

Chapter 1 - Introduction

Plants demonstrate a unique behavioral strategy for survival when compared to the survival strategies employed by animals. The primary motive of this project is to study the genetic behavior of plants while forging of light, which is an essential requirement for it to live. Since plants cannot move around or search for a better place to live when they have insufficient supply of light, their advanced cyber infrastructure allows them to pursue other techniques like germinating long before the actual period of germination, or altering the angle between the leaves, or growing taller in the vicinity of a plant that is of same height and is an equal competitor for light.

This study of plant's growth involves a huge number of plants (6300), with plants belonging to different genotypes (1050). All these plants are grown together so as to study the growth of plant belonging to a particular genotype in presence of a plant of differing genotype. This study has to guarantee that all plants receive an equal distribution of light. In normal conditions this is not guaranteed because, plants that grow near growth chamber walls have an advantage of not only getting the direct light rays from the light sources fixed above but also get light that is reflected from the walls, while the plants that are in the center or away from growth chamber just gets the light from above. In order to avoid this problem, plants are moved every couple of days. As already mentioned, a particular plant behaves differently when paired with different genes, it is important to know the plant with which a particular plant is placed every time there is a rotation.

In order to track this, every pot is given a bar code, which is used as an identification marker. But these are placed on the side walls of each plant and won't be captured in the cameras placed above. Moreover, barcodes have a couple of drawbacks when it comes to efficiency. Chapter 2 discusses more on these drawbacks. Therefore, Colored dot codes are used as identification markers. These codes are generated using specific colors and geometric shapes. These codes are then printed on vinyl sheets and glued to the plant pots such that they are easily captured by the cameras placed above. In the final step, decoding of each dot combination after image processing should provide information of a plant's position in a tray after every rotation of the trays containing the plants.

Chapter 2 - Requirement Analysis

2.1 Related Work

There are many kinds of experimental units that are being used in the field of image processing. Some of the prominent experimental units are Bar Codes, QR Codes, RFIDs, etc. All these were reviewed very carefully and few of them were experimented practically. Bar Codes and QR Codes were the initial choices. But these could not satisfy the purpose for various reasons. The pros and cons that were found during experimenting each of these have been given below in detail.

2.1.1 Bar Code

A barcode is a machine-readable representation of data that relates to the object to which it is attached. Barcodes are represented as data by varying the widths and spacing of parallel lines. Barcodes are read by optical scanners called barcode readers. In recent times scanners and interpretive software have become available on devices including desktop printers and smartphones. It is very economical and easy to generate.

The main disadvantage of using bar codes is that, bar code readers cannot read the code if the label is wrinkled, dirty or smudged. Since this project deals with the plants in pots filled with soil, there is always a chance for the soil particles to sit on these bar codes and mislead the way the bar code is decoded. The angle in which the image is captured also makes a difference. There were cases where a bar code was not read correctly because of a zoomed image or an image captured in a particular angle.

2.1.2 QR Code

A QR code is a two dimensional quick response codes that is gaining notability and popularity. QR codes are easy to use and versatile. The code itself stores huge amounts of information that is easily scanned and stored onto a mobile device. Many businesses are adopting this code as a means of marketing and as another way to attract customers to the internet for more information. Just like the bar codes, QR codes are economical and can be easily generated.

QR codes too have the same disadvantages as the Bar codes. QR codes are not perfect. If the code is any smaller than 1.25 inches, it could end up being unscannable. This amount of space was not available in the confines of this experiment. The problem worsens if the image has to undergo some kind of zoom-in or zoom-out. The smaller the QR code is, the less noticeable it will be, which may lead people to overlook it.

2.1.3 Radio-Frequency Identification

RFIDs use radio technology and can be programmed to convey a wide assortment of information to interact with RFID readers. RFID tags are electronic devices that are manufactured and programmed rather than being printed on paper. They can be used to track inventory like bar codes do, but they can be scanned without being within the line of sight of a reader and are not limited to being scanned one at a time. RFIDs can contain much more information than a bar code. The main reason for not using RFIDs in this project was because of the "Ghost tags" issue. The main reason for not using RFIDs in this project was because of the possibility of "Tag Confusion". Every shelf in the growth chamber contains about 24 trays and all these trays are placed very close to each other. If each tray contains at least one RFID, there is a possibility of RFID reader reading the tags that don't belong to it. It is also vulnerable to damage by water2.2 Requirement Specification

2.2. Requirements

2.2.1 Software Requirements

The following software requirements have been considered.

Operating System: Windows 7

Language: Python, Java

Tools: Eclipse Helios IDE, Portable Python-2.7.5.1, Fiji

2.2.2 Hardware Requirements

The following hardware requirements have been considered.

Space for Input: 5TB

Space for code: 300 KB

Fiji was used as an image processing tool. The features of Fiji are described in detail in the section 3.2.4. Codes executed in Fiji exist as plugins. These plugins are either bytecode class files or Jar files. These Jar files/byte code files were generated using “Eclipse Helios IDE”. After image processing, Python language is used to locate the objects and read its pixel values. All the python files were written using “Portable Python-2.7.5.1”. “Portable Python-2.7.5.1” is a python programming language pre-configured to run directly from an USB. It comes with a variety of packages and libraries like ‘NumPy’, ‘SciPy’, ‘PIL’ and etc.

2.3 Feasibility Study

2.3.1 Economic Feasibility

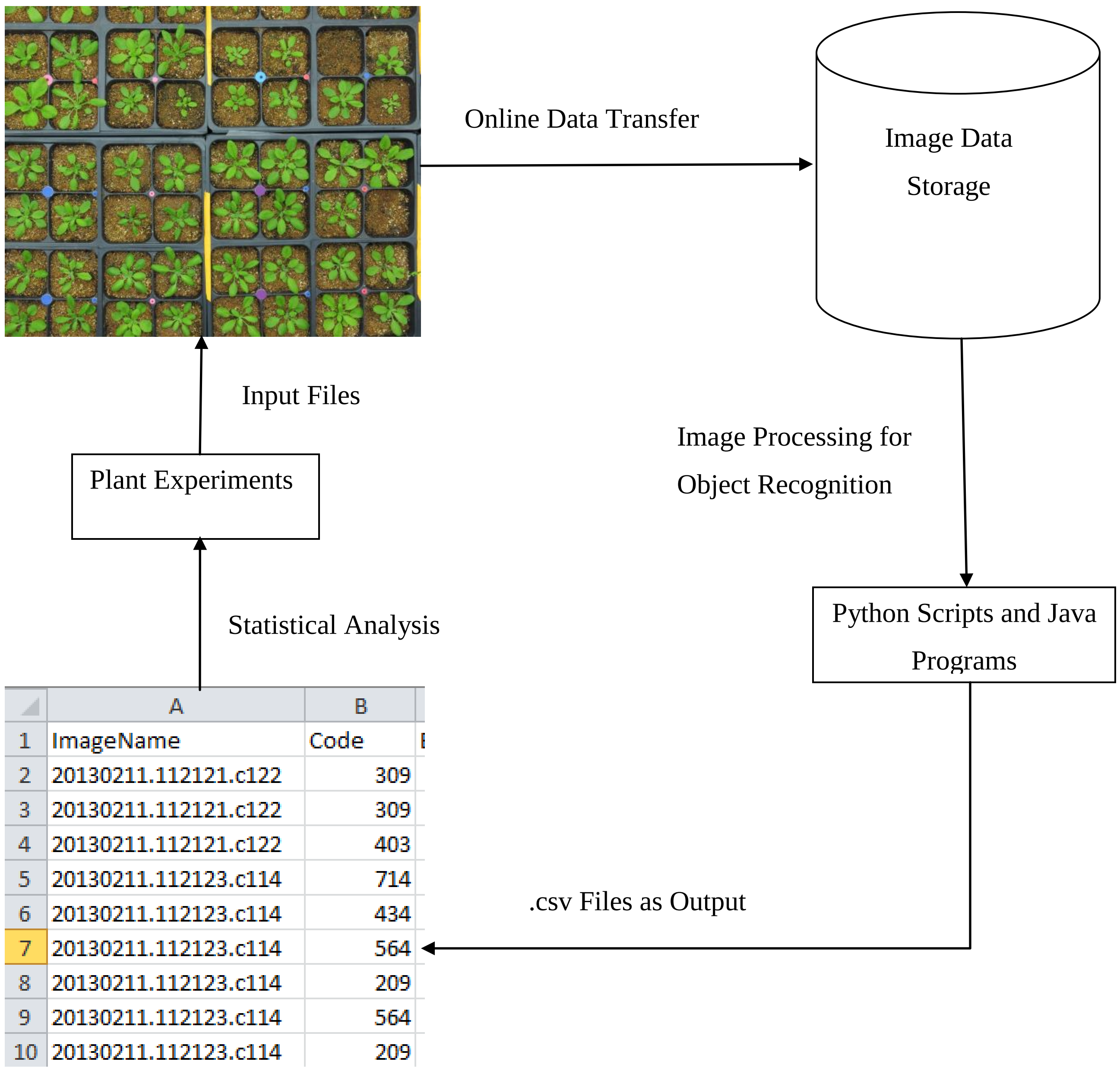
As far as processing is concerned, the application is economically feasible as it needs only Python and Eclipse platforms to be installed. Both Eclipse and Portable are freely available. For processing the images of one whole day, it takes 4 hours only. However, a server with a storage space of 5TB is needed to store 330000 images.

2.3.2 Behavioral Feasibility

Selecting the root folder that contains the images is the only thing to be done. Though this project involves various steps, all the steps are sequentially executed as all the multiple plugins used are called one after the other by a macro until the output is written.

Chapter 3 - System Architecture and Methodology

3.1 System Design



	A	B
1	ImageName	Code
2	20130211.112121.c122	309
3	20130211.112121.c122	309
4	20130211.112121.c122	403
5	20130211.112123.c114	714
6	20130211.112123.c114	434
7	20130211.112123.c114	564
8	20130211.112123.c114	209
9	20130211.112123.c114	564
10	20130211.112123.c114	209

Figure 3.1 System Architecture

The system architecture is shown in the above diagram. The first step in the design is the 'Plant experiments', which involves growing the plants and moving them on a timely basis. These plants are photographed once per hour during the 16 hours/day that lights are on and the images are stored in the servers. These servers are at the University of California at Davis. Automatically executing scripts transfer them to Kansas State University by way of an intermediate relay/backup site at the University of Texas (the iPlant Collaborative). All of these images undergo image processing. Java and Python languages are used for processing the images. The output of image processing is recognizing the identity of a plant and tracking its position in a shelf. This output is saved as a .csv file. The processes involved during each of these steps will be covered in detail in Chapter 4-'Implementation'.

A "growth chamber" is the place where the plant experiments are carried. The chamber set up for this experiment contains six shelves of plants, 108 cameras, HOBO® data loggers from Onset, Inc, fluorescent lights, LEDs and flats. HOBOs are the units for measuring humidity and temperature in the environment. Flats are the containers used for holding the trays for watering. A diagram of a plant chamber is given below:

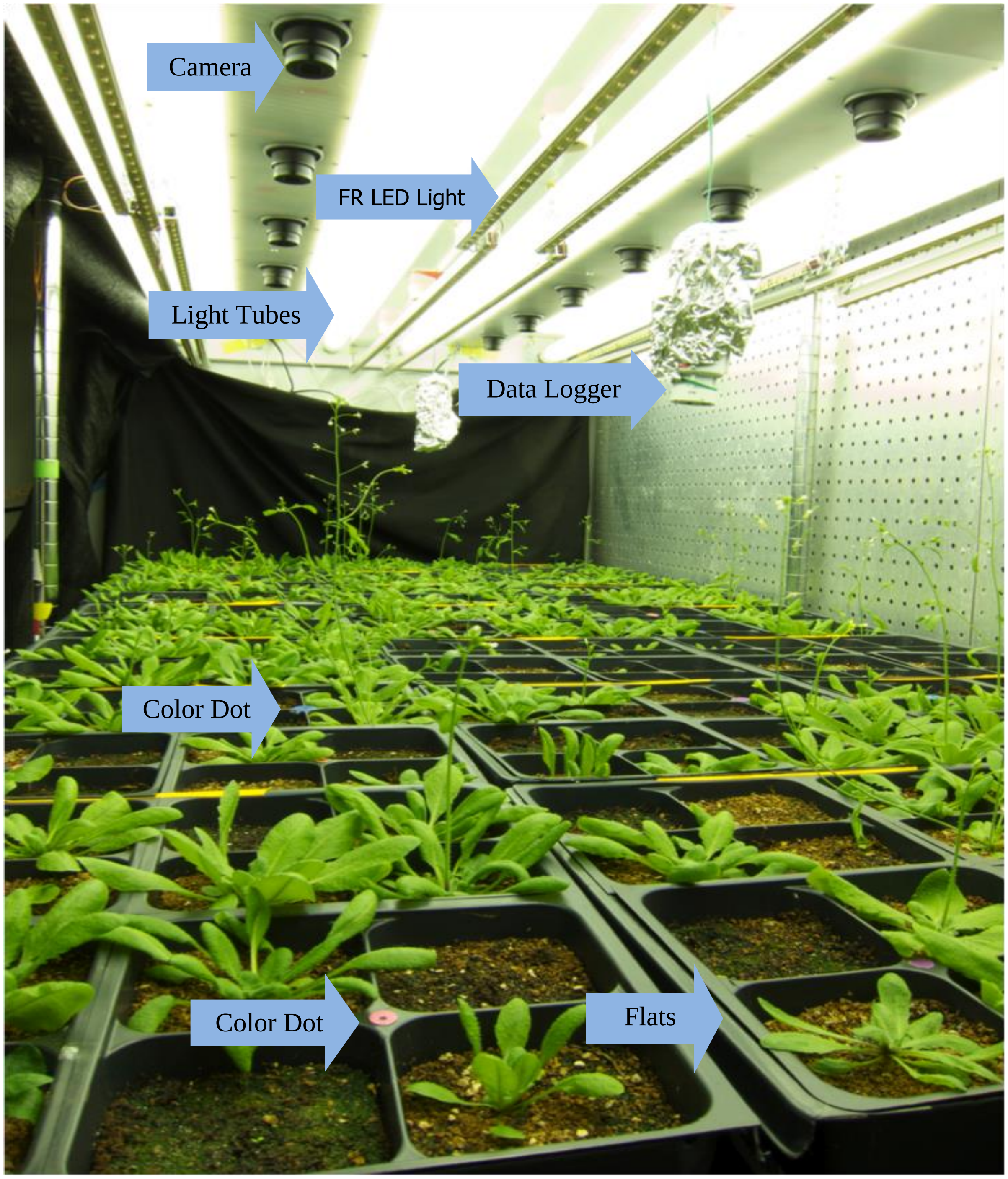


Figure 3.2 Experimental setup (See text for explanation of the arrows)

The above figure is a photograph of the growth chamber. (This picture was taken after the portion of the experiments requiring dot recognition was passed.) This is a picture of one of the six shelves used. The label ‘Camera’ points to the high definition cameras placed above the plants. Cameras are fixed in two rows of nine each. All plants are imaged by at least two cameras so, ultimately, they can be reproduced in 3D models. All these cameras are fixed. The label 'Light Tubes' points to the fluorescent light tubes placed next to the cameras. These tubes supply the light required for growth. However, the experiment requires controlled mixtures of light more reddish than fluorescent lamps typically produce. It is differences in the reddish part of the spectrum that enable plants to detect the nearby presence of competitors. In the above figure, the label 'RF LED Light' points to the LED lights used to supply the more reddish light. In the figure, the label 'Data Logger' points to a 'HOBO' that is used to measure the humidity and temperature in the chamber. The two labels called 'Color Dot' point to the color dots that are used for identifying the plants. 'Label 7' is the flat used for holding the plants in the trays. As seen the plants have grown long enough to cover the dots completely but this is not an issue as this photograph was taken when the code detection was no longer needed.

The main aspect of this project relies on image processing. A detailed description of Image Processing is given below. Other important concepts discussed here are Color System and Object recognition, which in this case is recognition of a circular shape. Detailed explanation of circular object recognition and color system is explained below. The methodology section also focuses on Fiji, software whose plugins and macros are extensively used in Image processing for object recognition.

3.2 Methodology / Concepts

3.2.1 Image Processing

There is a huge difference on how our own visual system works and how a task in image processing is performed. While the former seems to be easy to accomplish, there are lots of difficulties to be faced in the later. Trying to accomplish a robust, reliable and timely solution through image processing is almost impossible. Image processing becomes difficult when the task involves image analysis and not merely image enhancement or image editing. Image analysis means to extract or detect a pattern with respect to a computer's vision. The technology that is available today is not very helpful in this regard. Image processing technologies mostly concentrate on image editing which is more often called digital imaging. Some of the well-known software of this type includes Adobe Photoshop, Corel Paint, Gimp etc. All of these companies make use of technology that is fostered to accomplish the task of editing the image.

On the contrary, technologies like Fiji, ImageJ, Image-Pro Plus, MATLAB, ArcGIS, etc are used for image analysis.

In image processing, an image is described using a coordinate system. A coordinate system is imposed to get the position of an image element. Typically, the coordinate origin of the image lies in the top-left corner. X coordinates run from left to right and Y coordinates run from top to bottom. Rows and columns are numbered from 0 onwards. The size of an image is determined by the image’s width and length. There are different types of binary file formats that can used to represent and store image. Some of the formats that can be used are: TIFF, JPEG, GIF and PNG. All the images used in this project are TIFF files. TIFF is the Tagged Image File Format. It supports grayscale, indexed and true color images. It has the ability to support different images with differing properties. Using TIFF a number of variations of an image can be stored in different sizes and representations. For example, a single TIFF file may contain both a full-sized image and a thumbnail version of it. It is a universal exchange format that has made it to be widely used in archiving documents, scientific applications and digital video productions.

In scientific image processing principles an image is not just a photograph with a visual scenic beauty. Instead an image is a sample of information in an n-dimensional grid. In an image, a pixel is not something that just represents a color. A pixel is something that is reduced to a set of numbers. If a digital image is recorded then the values stored for a particular coordinate is not just a color value instead it is relative to the photon count.

3.2.2 Color System

In this project digital raster images are used. These are color photographs. All these images are represented as ordered arrays of image elements. The three primary colors are red, green and blue colors abbreviated as RGB. These are typically 8 bits per component. Each pixel requires 24 bits to encode all the three components of the colors i.e. 3*8=24 bits per pixel. Each color component has a range from 0 to 255. Towards 0 being the darkest and moving towards 255 is the lighter intensity. This project heavily depends on the color aspect of an image. Fiji depends on color images mainly in 3 ways: RGB images, RGB/HSB images and composite images.

For generating the dot codes, 5 colors are chosen such that they are distinctive in the presence of the color of rest of the plant data. Detailed description about the 5 colors used is given in Chapter 4. In spite of making careful choices and limiting the color range to be just 5 different colors, some problems occurred in distinguishing them. With time, some plants experienced stress and turned purplish. This may happen due to insufficient light or water or, in some cases, other atmospheric conditions. Due to the discoloration of the plants from green to purple, the color classification becomes a problem since one of the 5 primary colors that is used in this project is purple. Red and Pink were 2 other colors that were used for the dot codes. Problems occurred with these 2 colors too. At certain lighting conditions these two colors are hard to distinguish well from each other. Initially yellow was chosen as one of the primary color but due to its similarity in color with some leaves and certain particles used in the potting soil, this color was dropped. Green was obviously out of question as almost all the plant leaves have the color green. The pots and trays have the color grey and hence they too were not considered as one of the options. Finally, however the problem with the similarity between Red and Pink was solved by using a unique method of calibration. But for this calibration, all the pixel had to be in HCI (Hue, Chroma, Intensity) instead of RGB. For the conversion of the pixel values from RGB to HCI a simple algorithm is used.

3.2.3 Object Recognition

All the color dots used have a circular shape. The first step in the process of identifying the codes is to first identify a circular object in the image. This is called as Object Recognition. Object recognition in digital images has been an active area of research. There are many advantages in using geometric description for object recognition. There are a number of reasons why geometry has played such a central role.

- Invariance to viewpoint - Geometric object descriptions allow the projected shape of an object to be accurately predicted under perspective projection.
- Invariance to illumination - recognizing geometric descriptions from images can be achieved using edge detection and geometric boundary segmentation. Such descriptions are reasonably invariant to illumination variations.
- Well-developed theory - geometry has been under active investigation by mathematicians for thousands of years.
- Easy to make - a large number of tools are available for manufacturing these objects. In this project, a punching machine was used to cut the colored sheets into circular shapes.

3.2.4 Fiji

For the purpose of image processing, a tool called ‘ImageJ’ is used. This tool was developed by the National Institutes of Health. A further enhancement of ImageJ is Fiji. It is an open domain and provides a way of implementing the system design. It allows the user to view and interactively manipulate the images. Fiji is user friendly and allows the user to write scripts and codes in various programming languages like Python, C, Java, Jython, etc. Versions are available for the Windows, Linux and Mac OS platforms. Add-on codes written in Java can be used in Fiji/ImageJ by importing them as plugins. These plugins can be compiled and executed on fly in the running system. This aspect makes Fiji an ideal platform for developing and testing the code and algorithms efficiently. It is free software that can be easily installed. It is regularly used by researchers particularly in medical and biological imaging. Fiji comes with its own Java Runtime Environment so Java need not be installed separately in the computer. One of the key features of Fiji is the implementation of Macro language that can be used to implement large blocks of existing functions without the knowledge of the language Java.

Along with the compiler it also includes a built in editor. If needed, a Java program that has been compiled from another tool can be imported as a jar file. All that needs to be done is to place this plugin in a folder that is dedicated for such files. Through these plugins several tasks are implemented such as analyzing the problem, editing the image, processing the image, etc. Fiji can be efficiently used in areas radiological image processing, automated hematology image systems, multiple imaging and data comparisons, etc. This tool is extensively used for teaching and research purpose.

Some of the prominent features provided by Fiji are edit, analyze, process, print, save 8 bit color and greyscale images or 16 bit integer or 32 bit floating point images. A great many image formats are accessible in this tool. It supports mathematical operations like logical and arithmetical operations between different images, contrast convolution, Fourier analysis, edge detection of images, sharpening, smoothing, geometric transformation, scaling, rotation, flips and median filtering.

It includes additional packages like version control, issue tracker, dedicated development channels, rapid prototyping structure. Fiji can calculate area and pixel value statistics of user defined selections. It measures the distances and angles and creates density histograms and line profile plots. It supports contrast manipulation and does geometric transformations. It allows zooming up to 32:1 and down to 1:32. All processing and analysis are available at any magnification factor. Any number of images can be simultaneously opened. Spatial calibration is available even for units such as millimeters. Density and gray scale calibration is also available. Fiji supports four main essential freedoms defined by Richard Stallman in 1986:

- It provides the freedom to run the program for any purpose.
- Free Documentation is provided which helps in understanding how the program works and make changes to the already installed plugins
- Freedom for redistributing the copies
- There is a mail list available which allows users to submit queries and suggestions. They are given the freedom to improve the program and release improvements to the public

3.2.5 Macros

In order to identify the circular dots in the image a series of java plugins have to be used. All these plugins when put together and executed sequentially, help in object recognition The process of calling several plugins one after the other is made easy by being able to execute a series of Fiji commands. Fiji macro language contains a set of control structures, operators and built-in functions and can be used to call built-in commands and other macros. Macro code is stored in text files.

A macro is a small program that automates a series of Fiji commands. There is a way to create a macro by recording a sequence of commands using the command recorder.

3.2.6 Plugins

Fiji plugins are Java modules that extend the functionality of Fiji by using simple standardized interfaces. These plugins can be created, compiled and executed in any of the Java compilers. Fiji itself provides an option of compiling the plugins. Most of the Fiji functions are implemented as plugins. There are primarily 2 different kinds of plugins. One is the 'PlugIn' that does not require an image to be compiled and executed while the other is the 'PlugInFilter' that works only with an open image while executing the plugin. For these two plugins to work, two functions are created by default. The first function is 'setup()'. When the plugin is started, this function is used to verify if the capabilities of the plugin match that of the image that is being worked on. The second function is 'run()'. This does the actual work of the plugin. It is passed as a single argument 'ip', an object of the class 'ImageProcessor'. This contains the image to be processed and all the relevant information. It does not return any value but modifies the image and can create new images. The signatures of these functions are:

int setup(String arg, ImagePlus im)

The setup() returns a vector of binary flags that describes the plugin's properties.

void run(ImageProcessor ip)

The run() method returns no result value (void) but may modify the image that is passes and creates new images.

Chapter 4 - Implementation

The project is divided into 6 parts.

- Code Generation
- Pre Processing
- Poster Generation
- Color Calibration
- Color Classification
- Validation

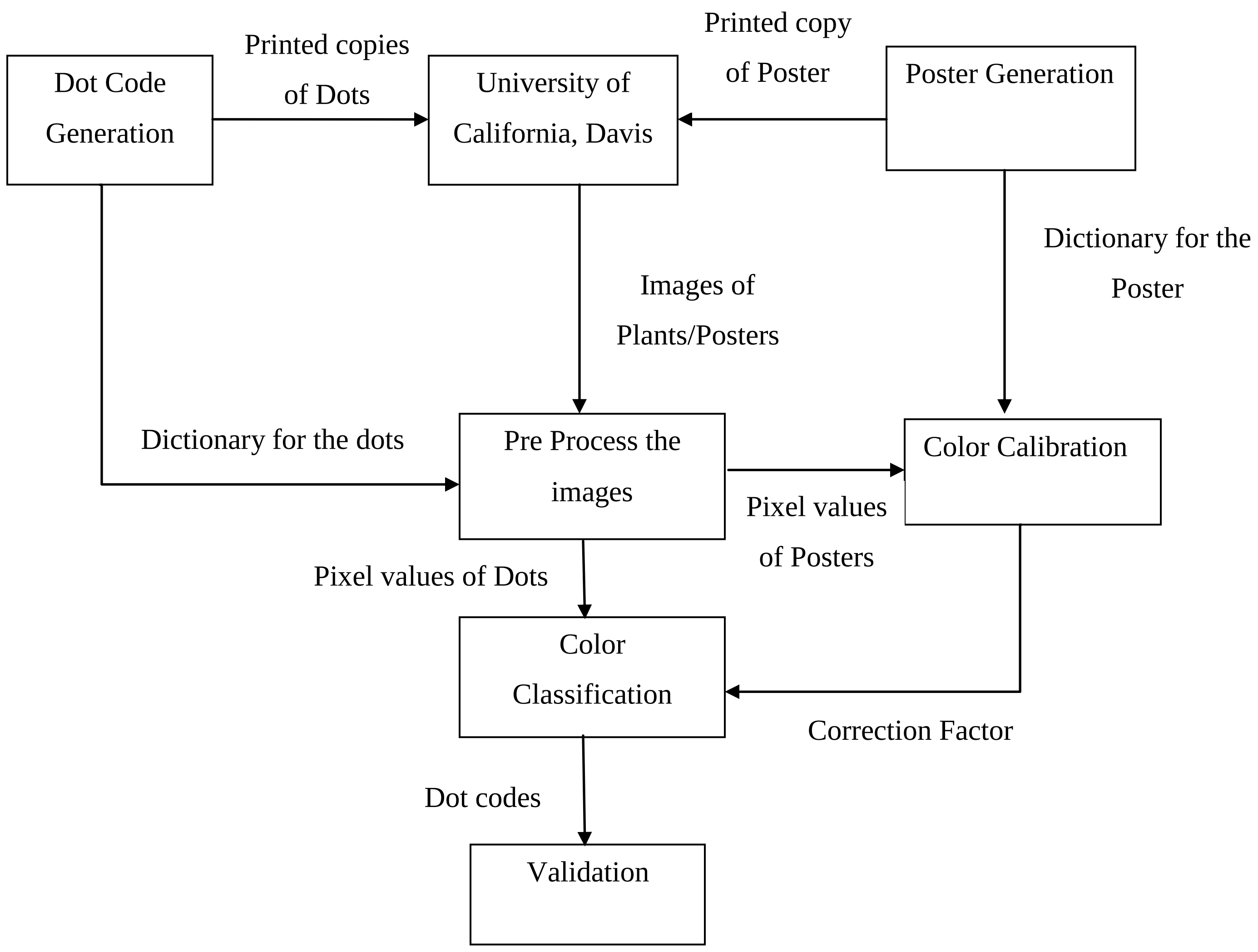


Figure 4.1 Block Diagram of the Work Flow

4.1 Code Generation

This is the first step in the project and this process involves generating color dots using 5 different colors. Colors are carefully selected such that they are difficult to confuse with the color of the soil, plant or pot data. Using the five main colors viz. Red, Pink, Dark Blue, Medium Blue and Purple, five additional "colors" are generated by adding a black dot in the center of the dot. In essence, the ten colors are Red, Pink, Dark Blue, Medium Blue, Purple, Black-Red, Black-Pink, Black-Dark Blue, Black-Medium Blue and Black-Purple. The dot codes attached to each plant tray have three dots in a combination. One dot is twice the size of the other two and defines the end at which to begin reading. This kind of set up is essential because of the risk of the tray being rotated and the color combination being displayed in the opposite direction. For example a color combination 'Red, Black-Purple, Blue' whose code is equal to '100' when read in the opposite direction becomes 'Blue, Black-Purple, Red' whose code is '200' would give a wrong value.

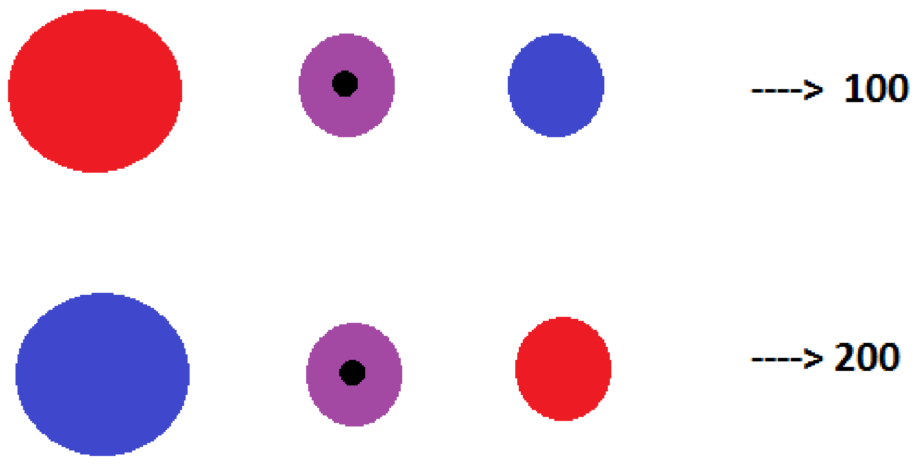


Figure 4.2 Different color codes

Early in the experimental design it was thought that perhaps the dots could also help with color balancing the image. For this reason a rule was formulated that no dot code group should have more than one copy of the same color. This rule remained in place after it was decided to use the color poster (described below) for balancing. Because there were to be 660 trays, it was calculated that permutations of ten colors three at a time were needed. Dot printing is done with a Python script using the imported packages 'pylab' and 'itertools'. Pylab is a plotting library and 'itertools' is used to iterate through the ten colors, permuting them three at a time resulting in 720 color combinations. Dot codes 661-720 are kept as a back-up in case of any damage to the dots being used. The 720 color dot combinations are printed onto a vinyl sheet. Vinyl is used because plants are frequently watered and so a waterproof medium is needed. The dots in the vinyl sheet are then cut into its respective shapes using a punching machine. These dots are then glued to the pot surface carefully so that they form the intended combinations. The circular shape was chosen for the dots because of the ease of cutting them using a readily available punching machine.

For the purpose of identifying the primary colors in the image a range of RGB values are specified for each of the 5 primary colors. The five primary colors used are:

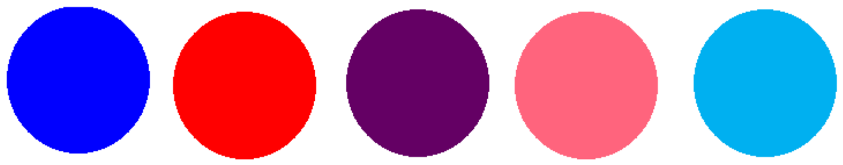


Figure 4.3 Five Primary colors

The hexadecimal values of these colors are: Red: '#FF0000', Dark Blue: '#0000FF', Purple: '#640064', Pink: '#FF647D', Medium Blue: '#00B0F0'. The first of the 3 dots is bigger and has a size of 1 inch. The middle and the last dot is 0.5 inch in diameter. The black dot that is at the center of few dots is about 0.25 inch.

Six shelves in a walk-in growth chamber are used in this project. Each shelf has 24 trays of plants. Each tray has 16 pots containing plants of different genotypes. Cameras are placed at a distance of 16 inches above these plants. These cameras capture the image of the plants once in every hour. Each image is named using the ‘current-date’, ‘current-time’, ‘shelf-number’ and ‘camera- number’. Current-time is saved as ‘YearMonthDay.HourMinuteSecond’. A typical picture is below:

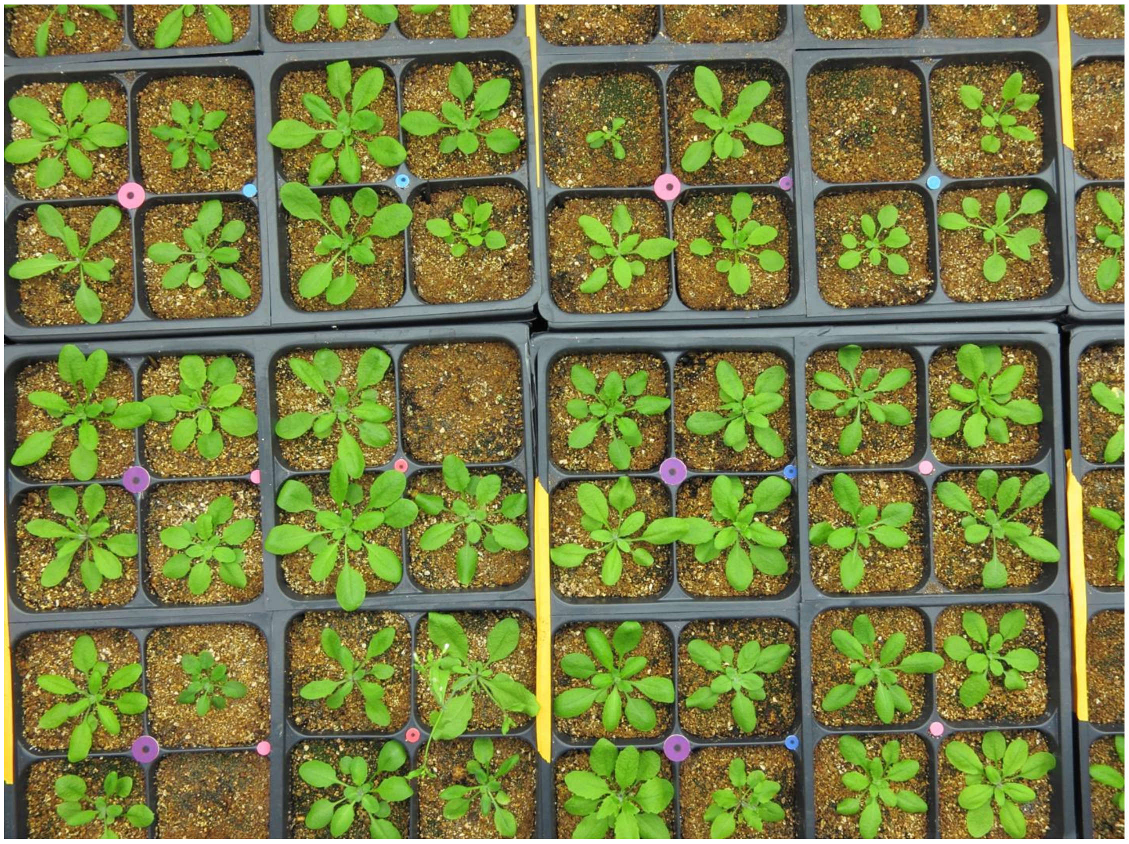


Figure 4.4 Plant image

The file name of the above image is ‘20130611.054023.c113.tif’. This nomenclature is read as follows:

- “20130611” stands for the date: Year-2013, Month-06 and Day-11.
- “054023” stands for the time: Hour- 05, Minute-40, Second-23
- ”c113” stands for shelf details: Shelf#-1, Camera#-13
- “tif” stands for Tagged Image Format.

Plants get their light from LEDs and fluorescent tubes placed right above them. Canon S95 cameras are used for this experiment. Each shelf has 18 cameras and all the 6 shelves together have 108 cameras. The aperture, focus and speed of the cameras used are F.71, 125 and 1/25secs respectively. The entire camera configuration is set up manually but operates under automatic control as programmed into each camera.. This camera has a CHDK framework and is powered by an adapter connected to an UPS to avoid any discontinuity of power supply in case of power outage in the lab/building.

The images are captured every hour for 16 hours per day. The experiment is replicated six times. Each replication is for a period of four months. All these together give a total of 330,000 images.

Separate folders are created for images taken at each hour. The name of this folder is the hour at which the cameras captured the images. Since the images are taken continuously for 16 hours, 16 folders of images are created per day.

The images captured are stored in 3 different servers to avoid loss of data. One copy of the images is saved in the Kansas State University Agronomy department, one is stored in University of California, Davis and one copy is stored in the cloud environment called iPlant Collaborative.

4.2 Pre-Process

Preprocessing is the second step. The images uploaded to the server undergo image processing to extract the circular dots present in the image. There are many steps involved in the process of recognizing the circular objects. In the initial steps the tool 'Fiji' is used and later part is done with the help of python code.

A macro is written that acts as a driver class and calls several other plugins that are needed to find the circular objects in Fiji. The only manual work that has to be done is to select the folder that contains the images in several sub folders and execute the macro, the rest all is carried on automatically. All the plugins that are called by this macro are written in Java language and are placed in the 'Plugins' folder of Fiji. Since there is no option to execute a Python code *directly* from Fiji, a Java program was written that will call and execute a Python code. This Java program is then compiled and called by the macro as one of the plugins. These plugins need to follow the usual Fiji standards, in particular to use the 'PlugIn' class or the 'PlugInFilter' class.

Eclipse was used to write the Java plugins even though Fiji itself has its own editor and compiler. The reason for using Eclipse was because of the efficient debugger that comes with Eclipse. Fiji plugins require an underscore to be included in their class names. Though this nomenclature is not a compulsory issue, it allows the plugins to be easily tracked. For example, any class that includes an underscore is always displayed in the drop down menu in Fiji. If the underscore is missing then the user has to manually navigate to the path where the plugin is saved. Fiji macros also have their own instruction set that is documented in the Fiji tutorial.

The preprocessing output files are stored in two folders. One folder contains the .csv files that has the details about the Area, “xy” coordinates and Roundness of the circles that have been detected while the other folder contains images showing the circumferences of the circles detected from the circular dots in the image. This folder is purely for reference/verification purpose. Within the folder intended to hold the .csv files, subfolders are created and they are named just like the input folders, having the name of the hour in which the images were taken. Inside each of these sub folders, .csv files are present. The macro file is stored as a plain text file in the ‘macro’ folder of Fiji. After the entire Preprocessing is done and the output files are written, the macro prints the number of files it processed. The steps involved in the Pre Processes step are:

- Color Segmentation
- Despeckling
- Gaussian Blur
- Remove Outliers
- Ellipse Detection
- Output Writing

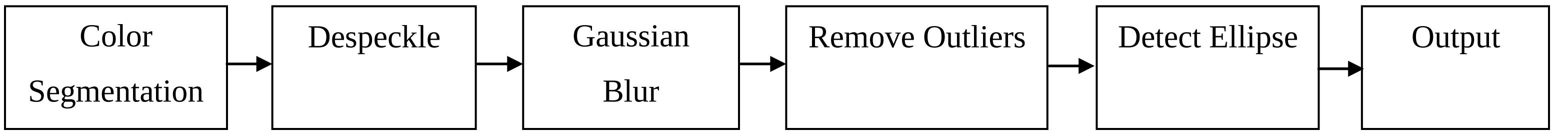


Figure 4.5 Block Diagram of Pre-Processing

4.2.1 Color Segmentation

This is the first plugin that the macro calls in the process of finding the circles. The first step in Color segmentation is to convert the color image to 8 bit gray scale image. This has to be done because some default plugins that are used as a part of finding the circular objects require the image to be a 8 bit gray scale image. Typically, the range of pixel values is 0-255 for 8-bit images and it is 0-65535 for 16-bit images. The command to convert to 8 bit image is:

```
<im.convertToGray8()>        // ‘im’ is the image
```

Because the dot color dots have been well chosen, this step removes virtually the entire image background but, at time, it can leave scattered pixels that do not belong to dots.

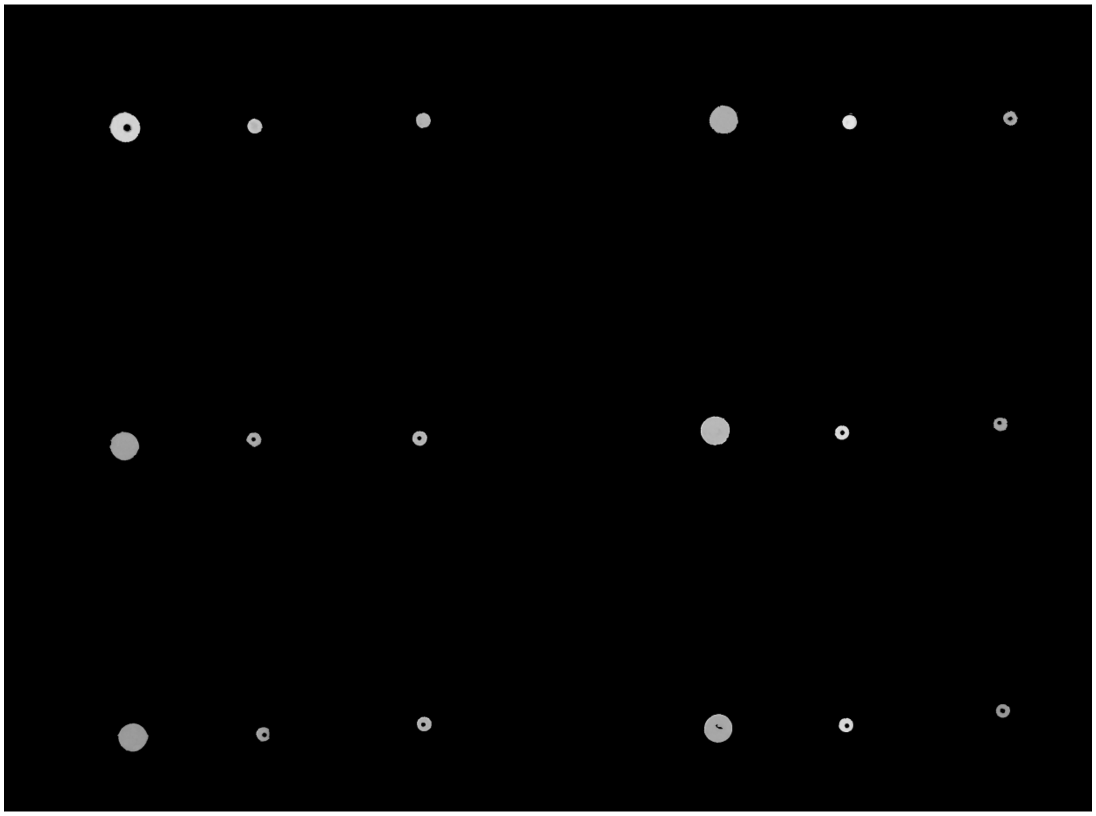


Figure 4.6 Color Segmentation

Some of the reasons for this are the tray color falsely appearing to be blue, or the flower colors in the plants being red and so on. Functions have been written to avoid this problem. . Despeckling, Gaussian blue and Remove Outliers are the functions used to avoid this problem. These are discussed in detail below.

4.2.2 Despeckling

This is the step that removes stray individual pixels that exist anywhere in the image that remain after color segmentation. These pixels which don't exist as a group of pixels are usually considered as noise. These pixels can be found scattered all around the image. The 'despeckle' plugin will remove almost all of these pixels. The below figure is an example of a non-despeckled image.

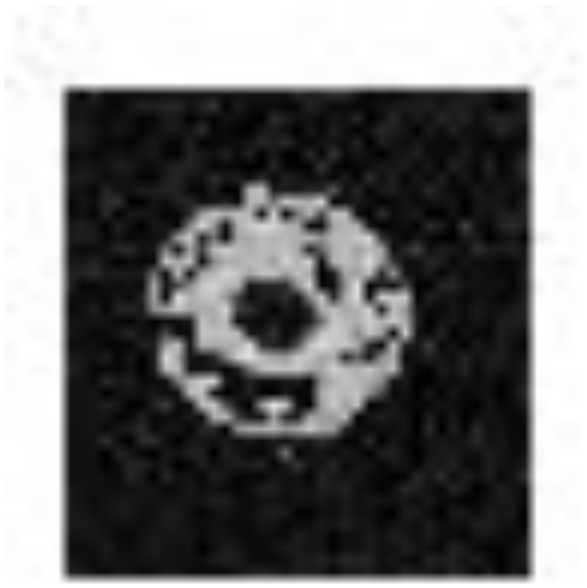


Figure 4.7 Before Despeckling: Image background is not clean

The below image is a despeckled image.

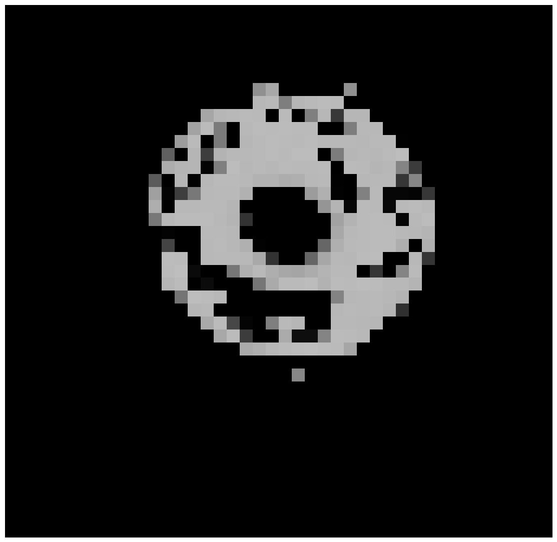


Figure 4.8 After Despeckling: Background is now uniformly black

Despeckling acts as a median filter. It replaces each pixel with the median value in its 3 x 3 neighborhood. This takes a lot of time because, for each pixel in the selection, the nine pixels in the 3×3 neighborhood must be sorted and the center pixel replaced with the median value (the fifth). Median filters are good at removing salt and pepper noise

4.2.3 Gaussian Blur

Gaussian Blur is used for correcting the circles whose details have been damaged by the presence of a sand particle or a leaf. By subjecting the image to Gaussian blur the image looks like as though it is being viewed through a translucent lens. It is basically used as a pre-processing step in order to enhance the image quality at difference scale.

During plant experiments, after a certain period of time the leaves grow big and tend to cover a part of a dot. This would not cause a problem so big so as to cover an entire dot because, by the time a leaf starts growing big enough to completely cover a dot (which would take almost 4 months), the next round of replication starts and the plants are replaced by seeds. Now, because the leaf covers a part of the dot, an entire circle would not be segmented in the color segmentation step. An example of this is given below.

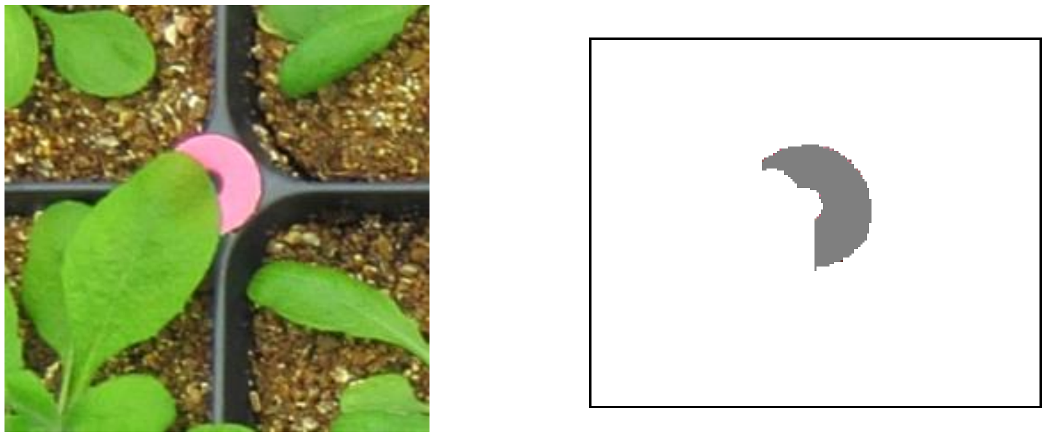


Figure 4.9 Before and After Color Segmentation

For such a partly segmented dot, an ellipse would not be detected as it does not form a complete circle although part of it has an arc like formation. When this image undergoes Gaussian function it gets blurred and when the blurred image’s threshold is varied a somewhat circular shaped object is formed. This is clearly shown in the below figure:

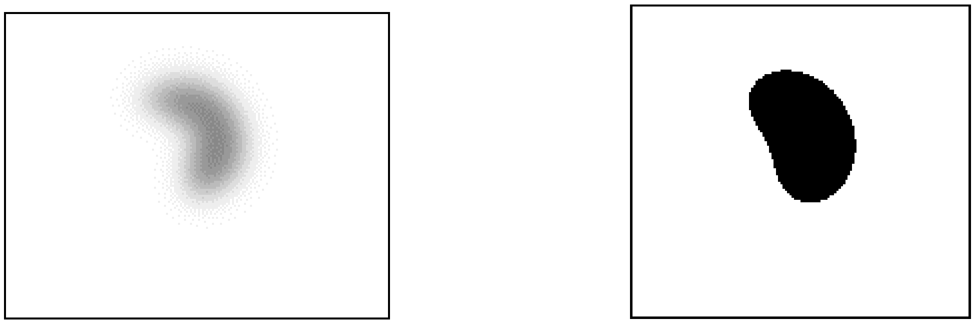


Figure 4.10 After Gaussian Blur and After adjusting Threshold

Object recognition plugin that is used in the next step is capable of detecting the ellipse from the above figure. This method can fail when more than half of the circle is covered by the leaf. The method that is described in the section 4.4.2-Check for Complications corrects such issues. Another example of such an instance is when soil particles are present on a circular dot. This too would lead to loss of data and if not for Gaussian blur, would lead to an unrecognized dot.

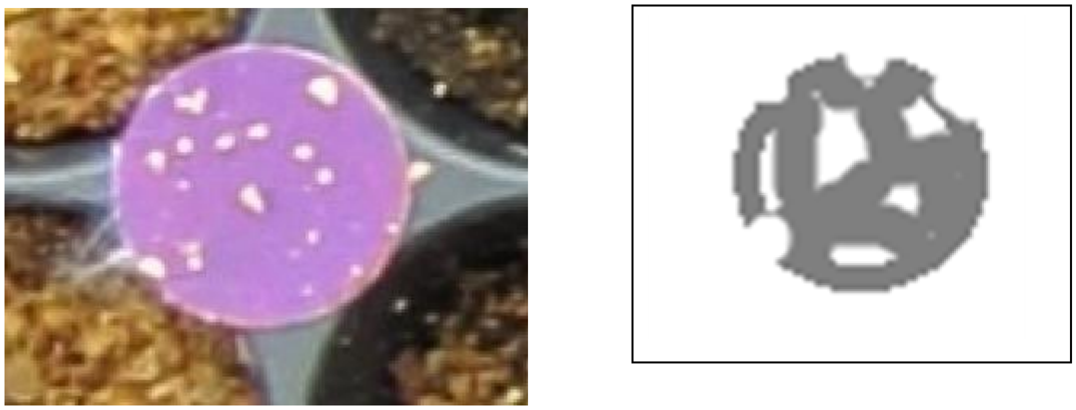


Figure 4.11 Before and After Color Segmentation

After Gaussian Blur the figure looks like below:

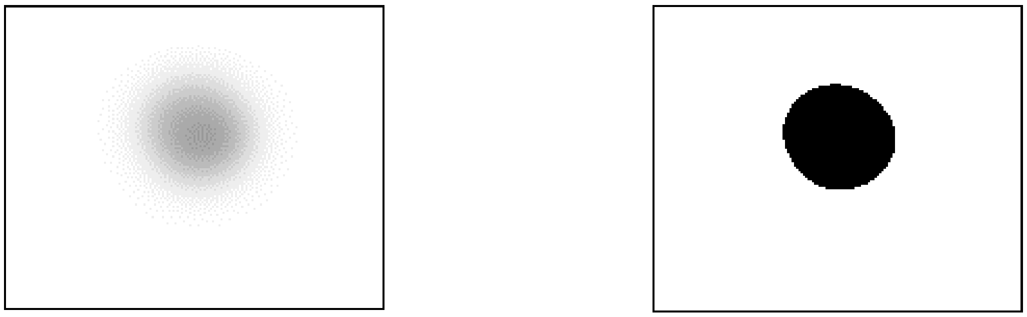


Figure 4.12 After Gaussian Blur and after adjusting Threshold

This filter assumes that out-of-image pixels have a value equal to the nearest edge pixel. This gives higher weight to edge pixels than pixels inside the image and higher weight to corner pixels than non-corner pixels at the edge. Therefore, when smoothing with very high blur radius, the output will be dominated by the edge pixels and especially the corner pixels. In extreme cases of a very high a blur radius, the image will be replaced by the average of the four corner pixels. For increased speed, except for small blur radii, the lines (rows or columns of the image) are downscaled before convolution and upscaled to their original length thereafter.

This filter's main function is for smoothing the image. *Sigma* is the radius of decay to

$$e^{-0.5} \ (\approx 61\%),$$

i.e., the standard deviation (σ) of the Gaussian, in which *radius* was $2.5 \times \sigma$.

4.2.4 Remove Outliers

This plugin is similar in nature to Despeckling. As in, it is used to remove the unwanted pixels from the image. While despeckling is used to replace each pixel with the median value in its 3*3 neighborhood, "remove outliers" is used to replace a pixel by the median of the pixel in a given distance (radius) only if for the pixels that deviates the median by more than a certain value (threshold).

All the pixels that exist alone in the image are considered to be noise. These pixels can be removed by using the "despeckle" function. But there are other types of "noisy pixels" which exist as a group of pixels. As a group, their number is large enough to be unrecognized by the "despeckle" function (which searches only 3*3 area). Usually such noise will be found as single lines. The below image shows a dot that was detected after color segmentation, along with the dot a noisy line is also detected.

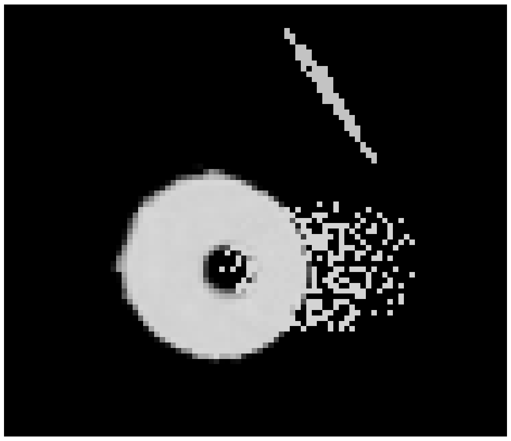


Figure 4.13 After color segmentation

Because the line in the image is formed by pixels that are placed close to each other the line won’t be despeckled. In order to remove such pixels, ‘radius’ and ‘threshold’ has to be provided as arguments. The “radius” determines the area up to which the pixels have to be searched and removed and “threshold” determines the kind of pixels that has to be removed. When the image is subjected to despeckling, despeckling succeeds in removing the noisy pixels around the circle as given below:

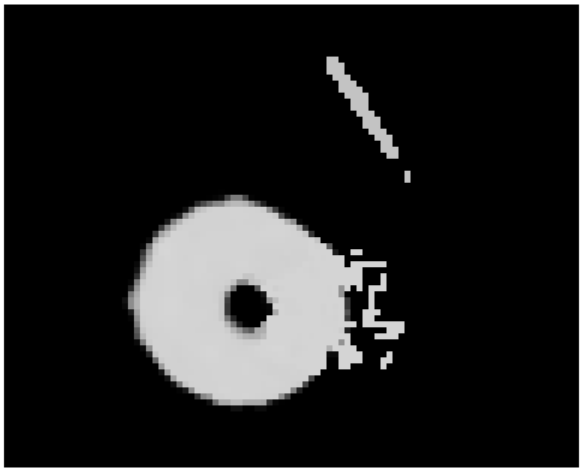


Figure 4.14 After despeckling

However, as it is seen in the picture, the line remains even after despeckling. Therefore “remove outlier” is called with the two arguments “radius” and “threshold”.

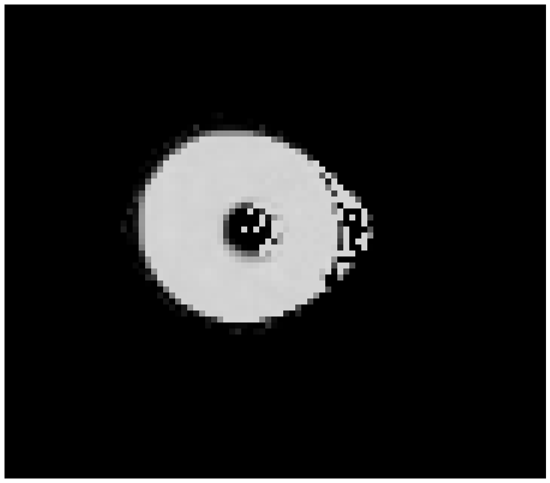


Figure 4.15 After remove outliers

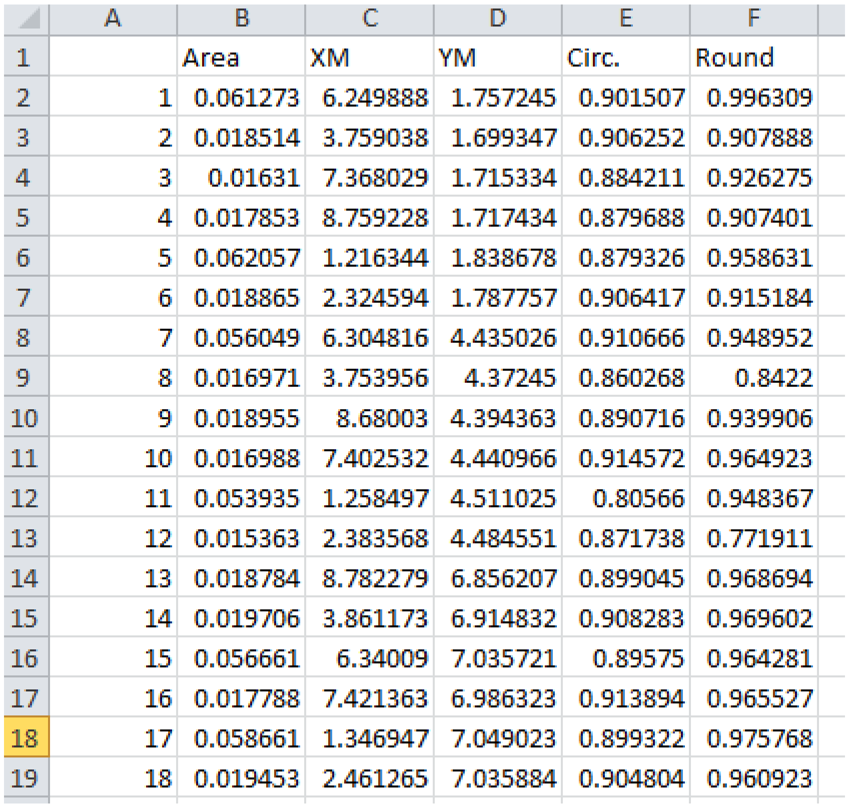
This succeed in removing the because of the absence of the pixels around the line and within the given radius.

4.2.5 Analyze Particles

This step is the one which finds the ellipses in the image. (Although the dots are all circular, they will appear elliptical except when photographed straight down.) After finding the ellipse it outputs a tiff image just showing the elliptical circumferences in a file with the same name as the input file in a new folder that is created for this purpose. This image is just for a reference. It can be used to manually check if the circles were detected properly. The main output is the file containing data pertaining to each circle. These values are the Area, “xy” coordinates and the Roundness factor of a circular dot. The Area of a circle is used to check if a dot is the beginning of a dot combination. As already mentioned, the dot groups are generated in such a manner that the size indicates which is the first to be read. “xy” coordinate denotes the relative position of the dots in the image. Roundness is used for a very important reason. Sometimes due to noise, non-dot particles with color similar to the dot, false circles are detected and 95% of these false circles are elliptical or ovular instead of a normal circular shape of a dot. For an ellipse which is ovular in shape, the roundness factor is very low. An ellipse that is completely circular has a roundness factor of 1 while an ellipse with an ovular shape has a roundness factor of >0.5. However, this is not always true. Sometimes a real circular dot will be detected as an ellipse with more ovular shape rather than a circular shape. These errors are taken care of in the next step which is section 4.4-Decoding.

The next step in particle analysis checks that the size of the circles is within a specified range. Sometimes due to many factors the soil particles are confused to be a circle and to avoid this, a range is provided to limit the radius of the circles to be within the radius of the small dot to the radius of the big dot. Circles that are either smaller than the small dot or bigger than the big dot are removed.

The output of this step is a set of .csv files that contain the details of all the ellipses that have been detected. Again the output that is written as a comma separated value with the same file name as that of the image name.



	A	B	C	D	E	F
1		Area	XM	YM	Circ.	Round
2	1	0.061273	6.249888	1.757245	0.901507	0.996309
3	2	0.018514	3.759038	1.699347	0.906252	0.907888
4	3	0.01631	7.368029	1.715334	0.884211	0.926275
5	4	0.017853	8.759228	1.717434	0.879688	0.907401
6	5	0.062057	1.216344	1.838678	0.879326	0.958631
7	6	0.018865	2.324594	1.787757	0.906417	0.915184
8	7	0.056049	6.304816	4.435026	0.910666	0.948952
9	8	0.016971	3.753956	4.37245	0.860268	0.8422
10	9	0.018955	8.68003	4.394363	0.890716	0.939906
11	10	0.016988	7.402532	4.440966	0.914572	0.964923
12	11	0.053935	1.258497	4.511025	0.80566	0.948367
13	12	0.015363	2.383568	4.484551	0.871738	0.771911
14	13	0.018784	8.782279	6.856207	0.899045	0.968694
15	14	0.019706	3.861173	6.914832	0.908283	0.969602
16	15	0.056661	6.34009	7.035721	0.89575	0.964281
17	16	0.017788	7.421363	6.986323	0.913894	0.965527
18	17	0.058661	1.346947	7.049023	0.899322	0.975768
19	18	0.019453	2.461265	7.035884	0.904804	0.960923

Figure 4.16 .csv file output of Fiji

The above figure depicts one of the csv files that has been output from the Analyze Particles step. The first field is just a list of serial numbers. Second field is the area of the circle that has been detected. At an average, the area is 0.05-0.06 for a big circle and it is 0.01-0.02 for a small circle. As seen in the figure values that lie within both of these ranges can be found. The next two fields are the X and Y coordinates of the centers of each circle. The fifth and sixth fields are the circularity and roundness respectively. Circularity is the measure of how close the dot is to being a complete circle. For the formula: 4π*area/perimeter^2, a value of 1.0 indicates a perfect circle. As the value approaches 0.0, it indicates an increasingly elongated shape. Values may not be valid for very small particles. Roundness is the measure of the elliptical behavior of the circle. For the formula: 4*area/(π*major_axis^2), a value of 1.0 indicates complete roundness. As the value approaches 0.0, it indicates an ovular shape.

The below figure is the second kind of output of this step. It is just a simple .tiff file that has only the circumference of the detected circle.

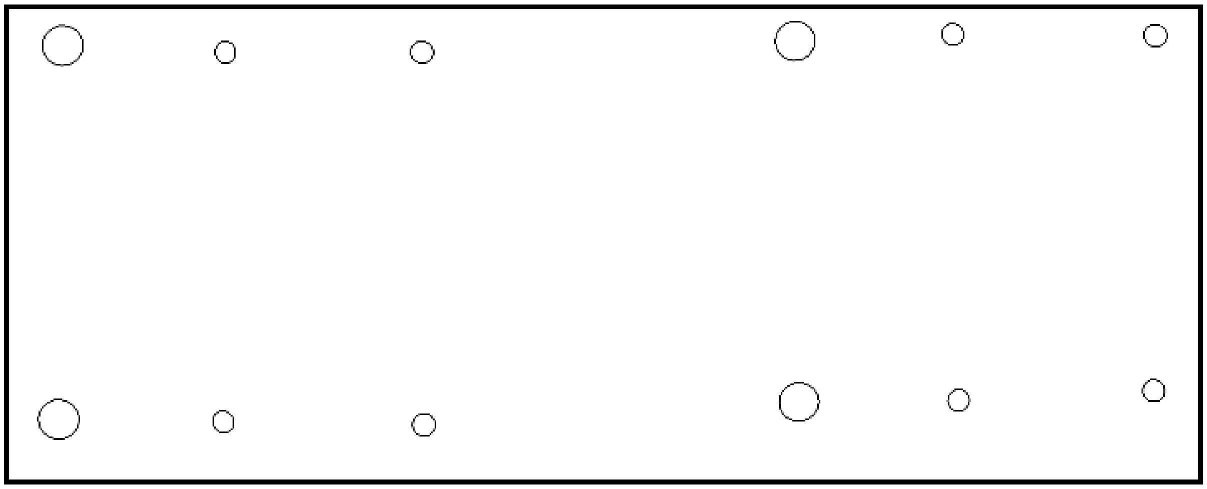


Figure 4.17 Ellipses of the detected dots

4.3 Calibration

Calibration is carried on to normalize the color of the dots in the images. The 5 primary colors selected initially do not retain the same RGB values in the images. Due to the influence of different kinds of lighting effect like the influence of incandescent light, fluorescent light or shadowing they appear to be different than their original color. For example, Red can appear to be orange, Pink can be quite close to Red, occasionally Purple resembles Dark Blue, while Dark Blue can be light enough to be confused with Medium Blue. This behavior is not universal among all the cameras, it happens for few of the cameras and it does not happen every time an image is captured. It is purely dependent on the environmental conditions that exist at a particular time in a day. Calibration is a step used in order to estimate the real true values of colors in the image.

Because this behavior varies from one camera location to another, a “normalization” or “correction” has to be calculated for each camera. This correction gives an estimate of how far the RGB value in the captured images differs from that of the RGB value of the original dot. The RGB value in the captured image are then adjusted so that all the dots in the image are closer to their original pixel values.

Considering the fact that lighting conditions change slowly if at all, calibration is carried out once per replication. In order to carry out this process a printed poster filled with rectangular blocks of known colors (called “bulk colors”) is photographed by all the cameras. The images of the poster from all the cameras are processed to find out the apparent RGB value of the bulk color of each box. These values are computer-compared to the known poster values. The differences are processed into normalization or correction formulas for each camera. The entire process of calibration is divided into five steps:

- Make poster
- Preprocess poster
- Identify all squares
- Get bulk color
- Compute correction method

Dr. Welch wrote code for sections 4.3.1 (Make Poster) and 4.3.3.1 (Introduction for reading a poster), 4.3.5 (Compute correction method) and, in the next section, 4.4.4 Classify color

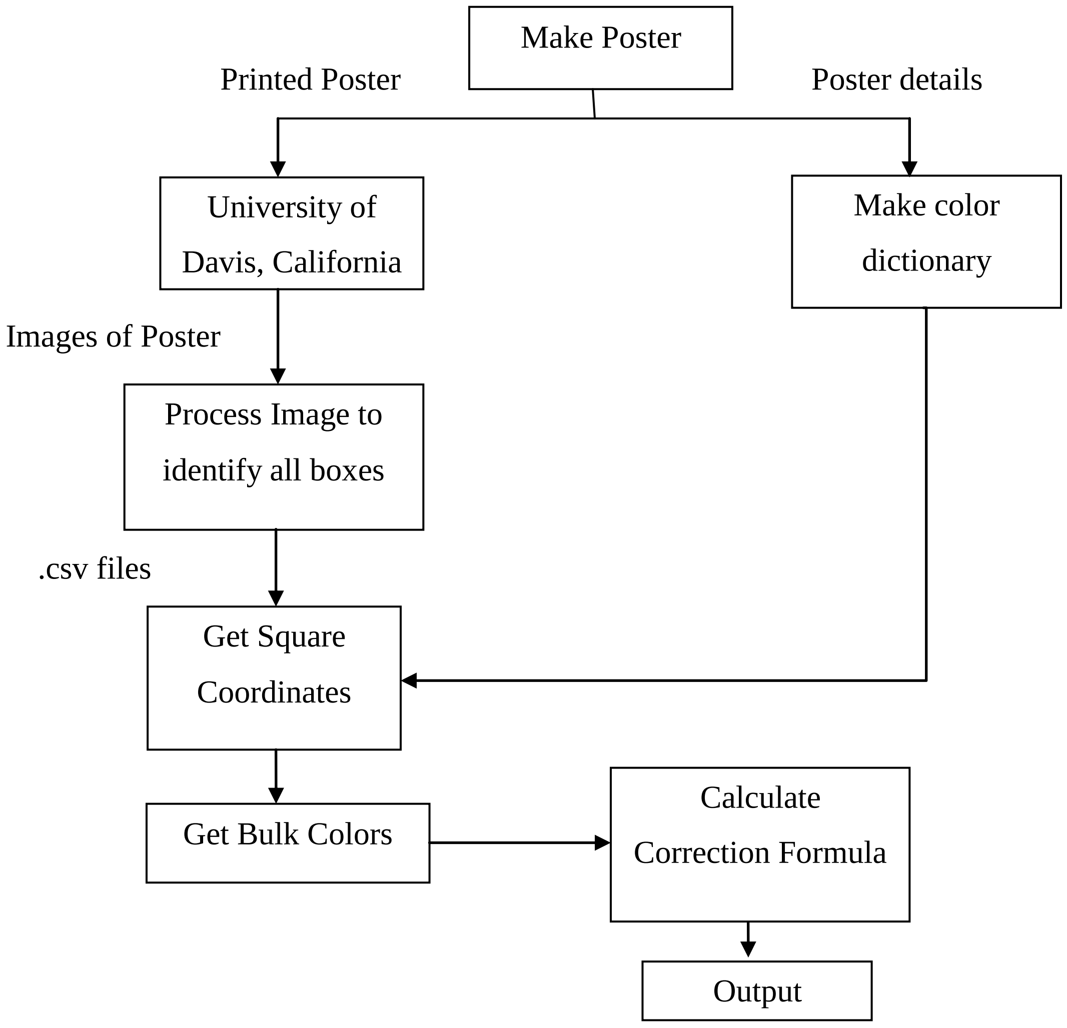


Figure 4.18 Block Diagram for Calibration

4.3.1 Make poster

As originally conceived, the poster had two purposes:

- To provide a coordinate system in which to locate camera positions;
- To aid in camera color calibration. Only the second step has proved necessary.

To create the poster Dr. Welch wrote Python code that produced a .png file describing a grid of 87 by 30 one-inch squares. Each square contains five parts. In the center there is a small color swatch containing a large gamut of RGB colors. Surrounding this is a larger square containing the bulk color of the square. Surrounding that is a thin black line that separates the bulk color from the border color. The border colors are pure Red, Green, or Blue arranged in a manner described below. Finally, a white line divides each square from its neighbor.

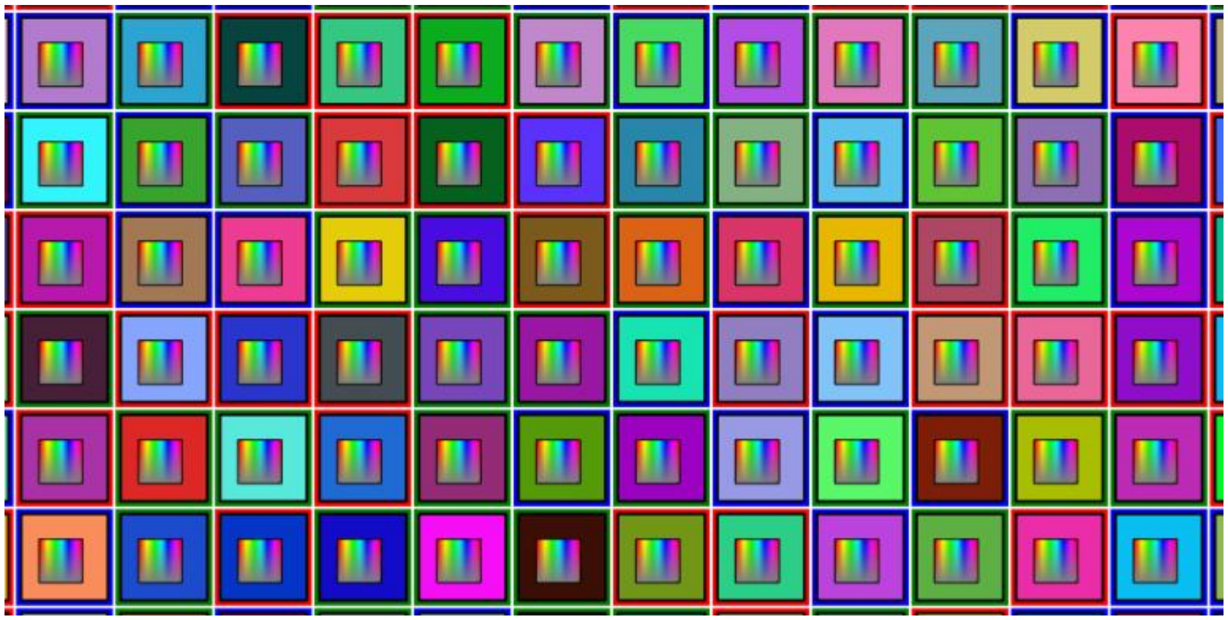


Figure 4.19 A small section of the poster

Also output is a csv file containing the RGB values for each square's bulk color (which was chosen randomly), The values for each color are a floating point number between zero and one. A final output is a JSON file containing a Python dictionary of information needed to uniquely identify each square when viewed in an image. This dictionary is described below.

The poster was then printed on vinyl, transported to the University of California at Davis where it placed on each shelf and photographed by the cameras at the start of each experimental run. It is important to emphasize that each camera only sees an area of approximately 15 by 22 squares in size. Additionally, at the time the poster was designed and created, it was not clear if all cameras would have the same orientation. It was considered possible that cameras might individually have any one of four cardinal orientations (descriptively called NEWS for "North", "East", "West", or "South"). The image processing steps described next handled these problems.

Once the poster is made, a dictionary is created with two kinds of keys. For the first kind of key box number is used as a key whose corresponding record is the bulk color of that particular box. For the second kind of key, a unique string of 25 characters is used and its record is the box coordinate. The details of forming the 25-character key are described in 4.3.3

4.3.2 Preprocess the poster

After the poster images are forwarded to KSU, they undergo the same kind of preprocessing as the plant images. Except that, instead of circular objects, rectangular objects are identified by Fiji. For every identified box the “xy” pixel coordinates and area are saved in a csv file. Similar to the plant images, an image that contains the detected boxes is also saved. The image file can be used as a reference.

Only the boxes with size within a specified range are recognized as valid boxes. Because a range is specified, neither the color swatch nor a group of boxes together is recognized as a box.

Other preprocessing steps for poster images include Despeckling, Remove Outliers, Gaussian Blur as were used for plant images. A few boxes may not be identified by the code. Functions are written in order to rectify this issue. These will be described in the further sections.

4.3.3 Identify all the squares

4.3.3.1 Introduction for reading a poster

In the preprocessing step, a csv file was generated that had the details of all the boxes that Fiji was able to detect. In this step, that csv file is read and the entries are saved into list. Fiji does not guarantee to put its findings the Morton scanning order and some squares may not be detected at all. These irregularities are detected and eliminated. In particular entries for undetected squares are added based on their relative positions.

Next, a box is selected at random that has at least 24 neighbors adjacent to it in a circular manner. By determining its row and column coordinates, the poster positions of all other squares in the image can be determined along with the camera orientation. Dr. Welch made the dictionary in the below described format. The color dictionary is keyed in two ways: one is a 25 character string that is unique to a box; the other is the box row and column coordinates. The first key type returns the box coordinate and the bulk color within the box. The second type will give the bulk color as well as the 25 character keys in all the four directions (NEWS). The 25 character string is called as a “signature” and it is unique to a box. This means to say, every box can be identified by its signature.

The signature is formed from the border colors of a group of 25 squares centered on the square of interest. As already mentioned, the boundary color can be Red, Green or Blue. These colors were chosen because they are sufficiently different that they should be recognizable no matter how badly colors are changed. The border colors were assigned at random with the idea being that a 25-letter signature would have odds of millions to one against being duplicated for any two squares in any of the NEWS (North, East, West and South) orientations. In Figure 4.20 below, to get the North signature of the square marked in white, the boundary colors are read from the beginning of the black arrow. For all orientations, signature construction proceeds from left to right and top to bottom. The first line has the colors Blue, Green, Green, Green and Red, abbreviated as BGGGR. The same thing repeats for the next 4 rows, finally making the North signature “BGGGRRRBBBGGBBGRGBBBRBRGB”. If the camera happened to be rotated 90 degrees to the left (i.e. the image was rotated 90 degrees right) this same pattern of scanning would give the East signature. Note that only the boundary colors of the box are used to form the signature not the bulk colors within the boxes.

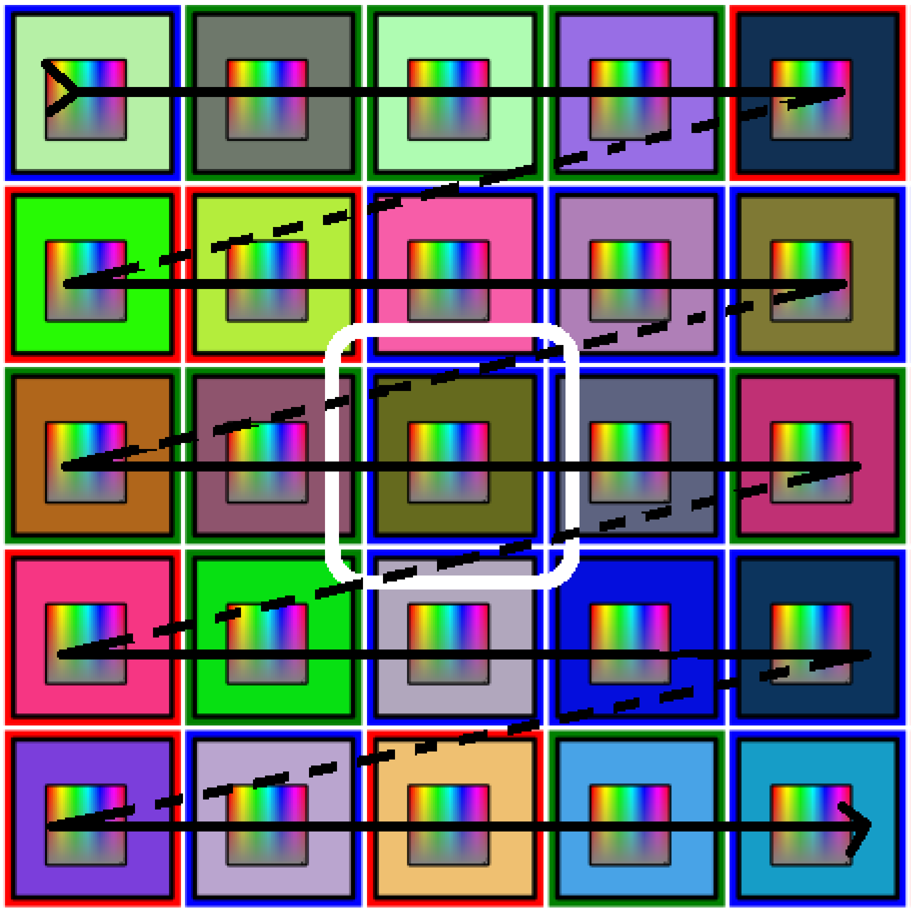


Figure 4.20 Morton scanning for reading the boundary colors

A typical record in the Dictionary looks like this: *"RGGGBBRRRBBBRRGRGRRGRRGBB": [[71, 3], [0.95465598399999996, 0.64093096199999999, 0.44099795000000003]],* where"RGGGBBRRRBBBRRGRGRRGRRGBB" is the signature key and [71,3] tells where the box is located in column (71) and row (3) coordinates. [0.95465598399999996, 0.64093096199999999, 0.44099795000000003] are the RGB values of the bulk color within the box. Therefore, once the signature of a square has been found, its box coordinates can be found. The box coordinates are the second kind of entry found in the dictionary. This record in the dictionary looks like this:

"(71,3": ["RRBBRRGBRGGRRRGBRRRGBGGBB", "RGGGBBRRRBBBRRGRGRRGRRGBB", "BBGRRGRRGRGRRBBBRRRBBGGGR", "BBGGBGRRRBGRRRGGRBGRRBBRR", [0.95465598399999996, 0.64093096199999999, 0.44099795000000003]]. The four signatures are placed in the order of 'NEWS'. Therefore, once any signature is found, the orientation of the camera can also be found. Once that orientation is known, the row and column coordinates of any square in the image can be found by counting squares in the proper directions. Doing so is the next step.

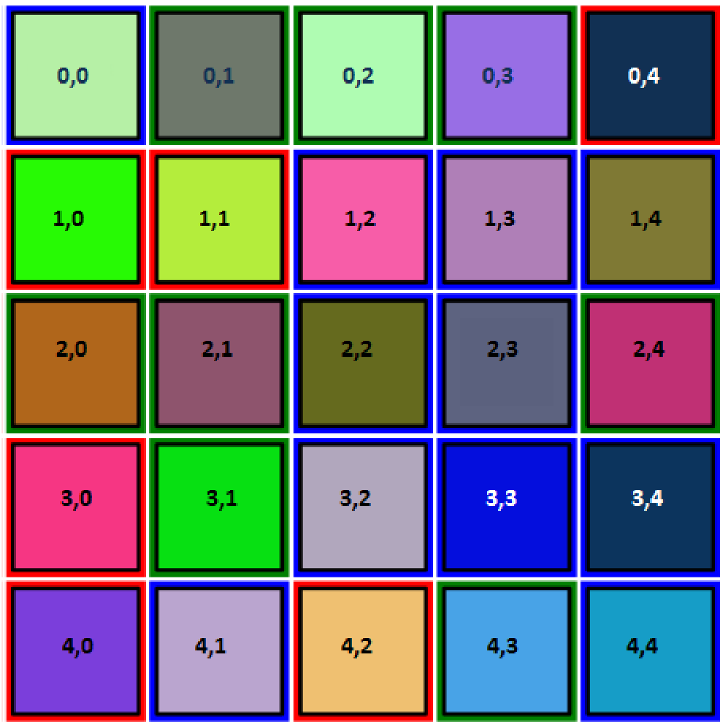


Figure 4.21 Poster section for naming the coordinate

In order to find the boxes in the images, Fiji tool is used. The plugin used here is to find objects that have a square shape. Only the square object whose size falls within a range is retained. Rest is deleted. The “xy” coordinates and ‘Area’ of all the detected boxes is written to a csv file. This csv file is again processed to filter all the very large and very small boxes. Due to noise in the image many of the boxes are not recognized and some extra false boxes are detected.

4.3.3.2 Method to read a poster

A white line divides each square from its neighbor (Figure 4.21). These white lines are used as a demarcation for finding the border colors of each box. That means any color that comes just before a white color is considered as a border color. So, the code reads all the colors in a single dimension until it finds a white color. It reads the color that is just before the white color as the boundary color of a box. But two kinds of issues are faced here, one is that these white lines don’t retain their original brightness (1.0,1.0,1.0 in RGB) in the captured images. It is due to the lighting effects and color bleeding. Color bleeding is caused due to the reflection of the neighboring color. The other issue is that the bulk colors of a few boxes are whitish in color. In the first case, the code goes on reading the pixels in the x direction to find a white color, and thereby goes past a box. This causes errors in reading the right signature of a box. In the second case, because the code encounters a white pixel even before it reaches a boundary, it reads a wrong pixel as the boundary color.

In order to avoid this, a counter is used to keep count of number of pixels read from one box to another. If it surpasses a range (approximate length of a box) then the reading stops there and a new box is fetched from the list and again the process of finding the signature continues until it comes up with a sequence of 25 characters.

Once the coordinate of one box in the image is fetched, the coordinates of all the other boxes is found by a simple counting process, iterating through all the squares that have been found by Fiji.

4.3.4 Get bulk colors

Now we have the center coordinates of all the boxes in the captured image, we have the box numbers of all the boxes and we have the original bulk colors of all the boxes. The next step to find is the RGB colors of the captured image. There can be noise in the color values of single pixels. Therefore, the average RGB value of about 30 pixels in each box is computed. This averaged color is then recorded next to the real bulk color of the corresponding box. So the output now is the ‘box number, real RGB value, image RGB value’.

4.3.5 Compute correction method

The final step is to develop transformations that, when applied to the dot colors in the images will make them more like those originally intended, therefore making them more readily classifiable. Dr. Welch wrote code to do this in the form of an object class called ColorProc which I made use of as described later. Methods in this class convert both the original and photographed bulk colors into a Hue-Chroma-Intensity color system in which the mathematical relationships between colors are more clearly described than they are in RGB. Then a set of curves are fit that describe the relationships between original and photographed colors for each of the three HCI components. These fits are quite good for the Hue and Intensity elements but less so for Chroma. This may be because Chroma is closely related to the concept of “Saturation” and is a more subjective element in human vision. A ColorProc method writes a database of these curves containing the results for all cameras. This database is subsequently used by other ColorProc methods to estimate original colors from RGB values extracted from photographs.

4.4 Decoding

This section describes how the codes which are in the form of 3-dot are finally decoded. The steps needed in this are:

- Assembling circle centers
- Check for complications
- Correct complications if any
- Average the pixel values
- Classify color
- Get the code that belongs to these three dots
- Write it to the output file

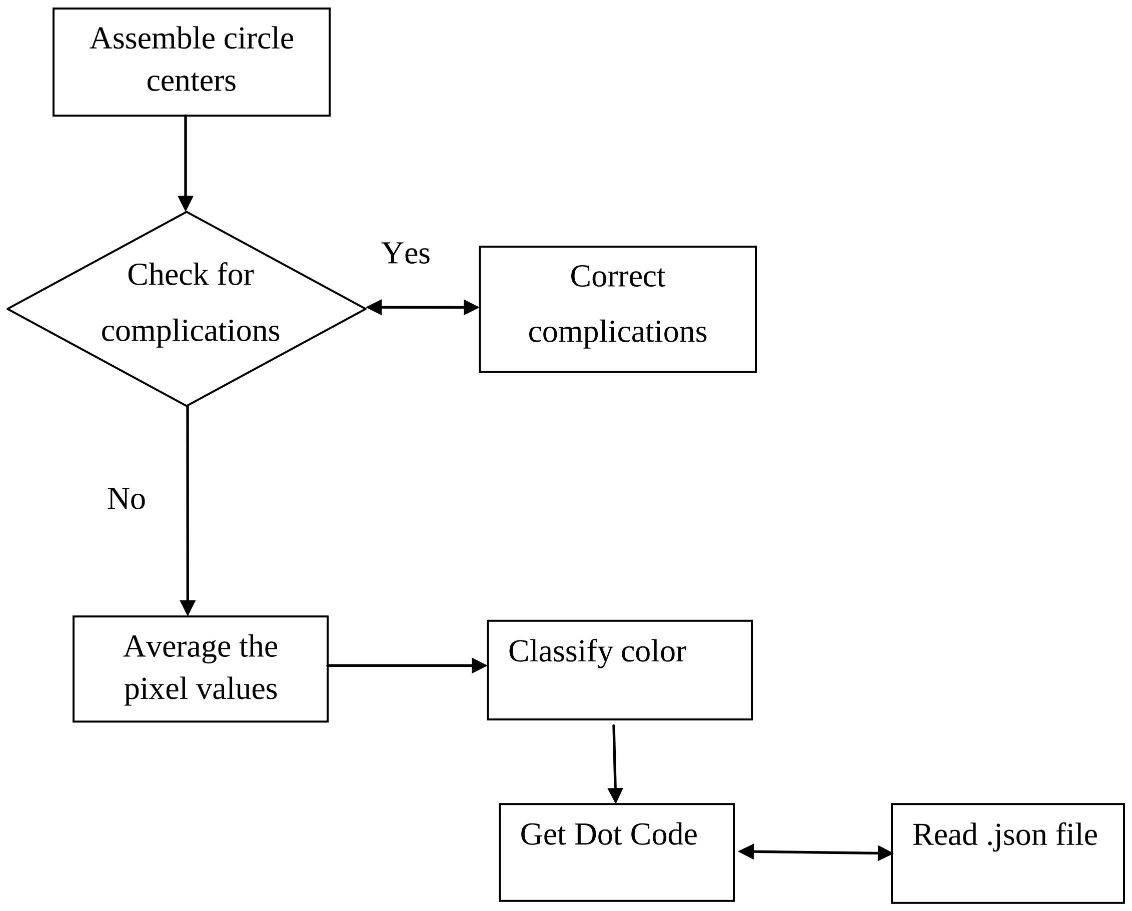


Figure 4.22 Block Diagram of Decoding

4.4.1 Assembling circle centers

After the preprocessing step a set of entries with "xy" coordinates, 'area' and 'roundness' is output to a csv file. A real dot has an area in the range of 0.05-0.06. This file is read and all the entries whose area is not within the given range are deleted. Also, the entries whose roundness factor is below 0.5 are deleted. These values are deleted because they usually belong to false circles. Once the obvious false circles are deleted, all the circles in the list are sorted such that all of circles fall into their respective rows. This has to be done as "Fiji" detects the circles in an irregular manner without following the Morton scanning rule.

4.4.2 Check for complications

The sorted circles are now checked for other complications. In the experiment it has been guaranteed that each camera is focused such that it either captures an entire dot code (combination of three circles) or it does not capture any code. Therefore in every picture the total number of circles is always in multiples of three. The distance between the circles in a code (distance between first-circle, middle-circle and last-circle) is also maintained constant for all the pots. The complications that can occur in this step are of two types:

- Loss of a true circle.
- Big circle being detected as a small circle.

The first case, that is loss of a true circle is caused when a leaf would have covered a part of a dot making the circle look too elliptical (such circles are deleted as their "roundness factor" would be below the normal range). This complication is revealed when the number of circles present in a row is not a multiple of three as mentioned above. The second case, is due to over filtering in the "remove outlier" step. This complication is recognized because of the fact that there should be at least one big circle for every three-dot code.

Using the fact that the distance between the circles in a code (distance between first-circle, middle-circle and last-circle) is always maintained constant for all the pots, the position of the missing dot is calculated using the position of the detected dots. A new circle ("xy" coordinate) is created and is plugged in the missing position. This solves the first issue.

If all the three dots in the sequence found within a length range is detected as small then the second issue arises. In order to fix this issue, one of these three small dots has to be made big. But because the three-dot combination could be read either from left or right, there is uncertainty as to which dot to be made bigger, either the first of the three-dot or the last of the three-dot. For solving this issue, the position of the other big dots in the image is fetched. If the big dot for the other three-dot combinations comes to the extreme left of the three-dot combination then the direction of the three-dot combination is noted as left-right, else the direction is noted as right-left. If the direction is noted as left-right then the first of the three-dot combination is made big else the last of the three-dot combination is made big.

4.4.3 Average the pixel values

Once all the complications are resolved, the dots are grouped into threes and pixel color values of these dots are averaged. Because there are two sizes of circles and dots may have a black portion in the center, the number of pixels that have to be averaged for different dots varies. Every circle has an outer portion and an inner portion. The inner portion is the portion that covers the black part if one is present and outer portion is everything in the circle other than that.

For a given center coordinate “xy” and a radius, all the coordinates that fall into a rectangular shape is generated. These coordinates are then checked for the formula of equation of circle. Suppose ‘HK’ is the coordinate that falls within the rectangular shape then only those (HK) values that satisfies the below equation are retained, rest of the values are deleted.

$$(X-H)^2 + (Y-K)^2 <= r^2$$

In the “color segmentation” step, segmented images were separately saved in a folder. Because this image is color segmented, it contains only those pixels that bear the color of the five colors used for the dots. Using this image for reading the pixels avoids the chances of including pixels that are not a part of a dot code. A leaf or a soil particle on a dot is an instance where these kinds of issues are faced. This image is used for averaging the pixel values of the coordinates calculated from the above equation. If the pixel value is a black color (due to noise present on the dot surface) it is ignored. Then the average pixel color is calculated for all the non-black pixels. Finally the color of the averaged pixel value is classified as described in section 4.4.4.

The next task is to find the pixels that are a part of the inner dot. The original, unsegmented image is used for this purpose. For the same center coordinate but with a smaller radius, all the coordinates that might form the inner dot is calculated and their pixel values are averaged. Sometimes, because of improper use of the punching machine, the black dot that should be in the center of the dot can appear to have drifted toward the circumference of the circle. In order to find the central black dot within a color dot, multiple inner circles are sampled. Hence at least five samples of inner circles are required to confirm the presence of a dot. The code is written such that, even the occurrence of one black in one sample out of the five samples, would confirm the presence of a black dot in the whole color dot. Therefore, up to five, differently positioned inner circles are sampled and averaged. If any of these averages fall below a specified threshold of intensity, the presence of a black dot is confirmed. The letter ‘B’ is then concatenated with the color of the outer circle. Otherwise, the color of the outer circle is taken to be the color of the dot.

4.4.4 Classify color

This step is to utilize this information to adjust the dot colors to be more like those originally intended; thus, to make individual dots readily classifiable; and then to classify them. Dr. Welch developed code to carry out these steps and I integrated his results as described below. These functions are carried out by three object classes: ColorProc and LineMan, provide respectively, color conversion and classification services to their client, DotClassifier, which makes dot color decisions. ColorProc contains methods which, in an off-line mode, analyze the outputs of the poster processing software to determine how RGB values, in general, have been changed by printing, chamber illumination, and photography. ColorProc also has a method, “CameraToTrue” which attempts to invert these changes to the extent that that is possible. DotClassifier creates a ColorProc object to access this service for each camera. Internally, ColorProc uses a Hue-Chroma-Intensity (HCI) system which is better than RGB for color description.

Unfortunately, Chroma changes are harder to reverse than Hue and Intensity because they have to do with more subjective aspects of human vision (i.e. color saturation). Therefore, rather than being exactly converted to their original colors, dots of a single color tend to make linear clusters in RGB space after conversion. The LineMan supervised learning class was created to cluster such data (see figure below) and enable classification of linear clusters.

LineMan operates in a manner analogous to K-Means but rather than try to place a point in the middle of each cluster, it tries to position one or more line segments. As with ColorProc, this learning takes place off-line to the dot analysis program with the results being stored. The learning is done via the basinhopping.py optimizer included with scipy 0.12.0. After DotClassifier asks ColorProc to correct the colors, it asks LineMan to determine which line segment in RGB space the corrected point falls nearest. This becomes the declared color. The signature for classifying the dot color is:

Object = ClassifyDotColor(self,dR,dG,dB,Cname,Convert=True):

Here dR, dG and dB are the R, G and B colors in floating point. Cname is the camera name, this argument is extracted from the image name. The output the ClassifyDotColor method is <True, ColorName>. Here True denotes success and color name is one of the 5 colors used. ColorName is the color of the outer circle.

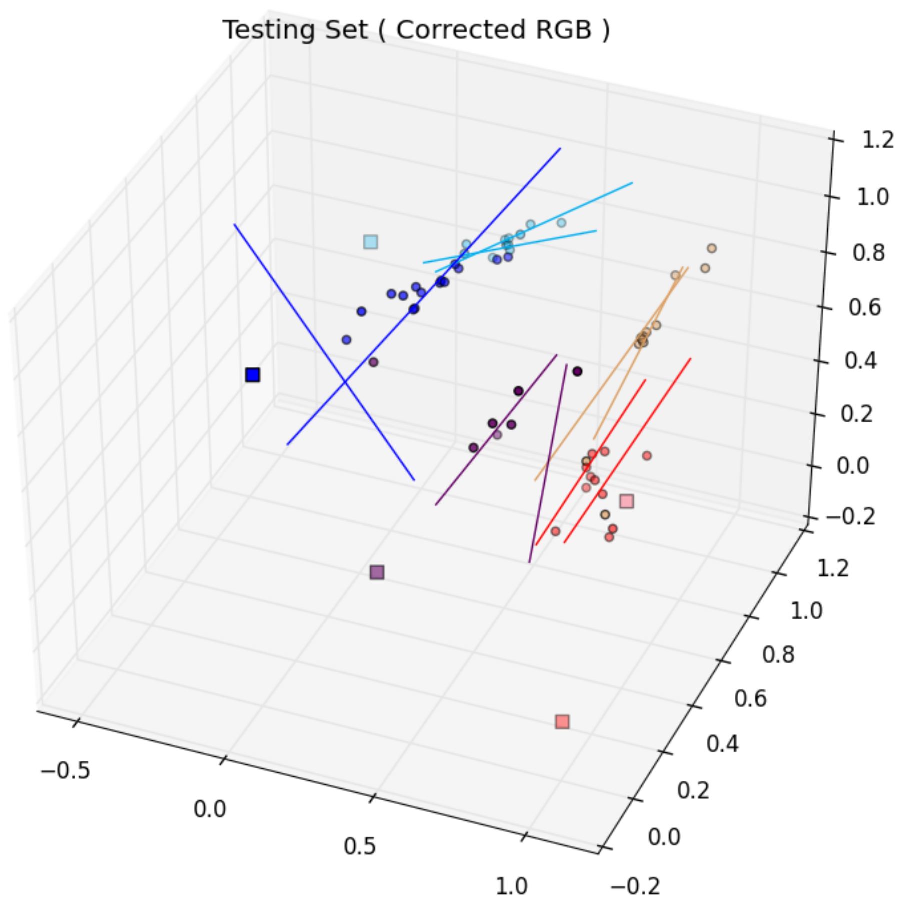


Figure 4.23 Color Classifier

4.4.5 Get the code that belongs to these three dots

As already mentioned, a json file is maintained that has the dictionary for the dot codes. It is created when the dots were generated. In the dictionary, the color combination for three dots is the key and the code for each of these combinations is the value. After the colors for each of the three-dot combinations is obtained in section 4.4.4, the json file is searched for this color combination.

Finally the code for all the three-dot combinations in a particular image is output to a file along with the “xy” coordinates (location) of these dots. And the same is repeated for all the other images present in the folder.

Chapter 5 - Testing

In Unit Testing, each module is tested as a separate module without the interference of other modules. This ensures that all the modules are error free and perform correctly. The entire unit testing was conducted manually.

Sl No	Section No	Test Case	Expected Result	Result
1	4.1	Generating the experimental units	Generate a set of unique 720 three-colored combination	Pass
2	4.2.1	Segmenting all dot colors from the non-dot colors	An output image for each input in which all of the dot colors are color segmented	Pass
3	4.2.2	Remove noise from the image	All the pixels in the image that do not belong to the dot colors are removed	Pass
4	4.2.3	Fix incomplete dots	Detect all the partly damaged dots that were not detected due to loss of information	Pass
5	4.2.5	Fetch the circles	Get the area and coordinates of the circle	Pass
6	4.3.1	Make the poster	A poster with 30 rows and 87 columns is created	Pass
7	4.3.2	Fetch the squares	All the squares in the	Pass

			posters are detected and their coordinates are calculated	
8	4.3.3	Identify the squares coordinates	Get the row and column numbers of all the squares	Pass
9	4.3.4	Get the square color	Fetch the averaged color of every square	Pass
10	4.3.5	Compute the color correction method	Get the relationship between the real color and the photographed color	Pass
11	4.4.2	Fix the complications	All the missing circles are detected	Pass
12	4.4.3	Get the circle color	Fetch the averaged color of every circle	Pass
13	4.4.4	Get the real color	Fetch the color of the circle which should be one of the five selected colors	Pass
14	4.4.5	Identify the code	Fetch the code for the three-dot combination	Pass

Table 5.1 Unit testing

Chapter 6 - Conclusion

This project aims at creating an experimental unit that is essentially color based and geometrical. This experimental unit guarantees some of the basic aspects like low cost, robustness, flexibility, durability, compatibility and fast.

Firstly, object recognition and color calibration guarantees that all the codes are correctly detected after image processing. The validation and verification done at the end of each stage is a proof for this.

Secondly, the only expenditure for using color dots as experimental unit is a vinyl sheet. These codes are printed on vinyl sheet and cut into circular shapes. This is very cheap when compared to the expenditure that goes into using scanners for detecting RFID's and Bar codes. RFID's which one of the best identification objects, requires batteries which are costly compared to the low cost vinyl sheet.

Thirdly, it is flexible in its area of use. The use of these color codes is not limited to just experiments on plant data. They can be used as an experimental unit in various areas of scientific research by using colors of choice.

Fourthly, it is durable. It does not have to be maintained with care like an RFID that has the risk of getting spoilt in the presence of unfavorable experimental conditions especially in this project where the plants are watered daily. Also, since it is not battery powered it does not deplete soon and can be used for a long time.

Finally, this method is comparatively fast. The total time spent in detecting the code for the three-dot combinations in each image is less than half a minute. Although the time spent in getting the bulk colors of the squares in the poster is 15 minutes per image, it happens only once per replication.

Chapter 6 - Future Work

As previously mentioned, every pot is given a bar code, which is used as an identification marker. But these are placed on the side walls of each pot and won't be captured in the cameras placed above. However, after every rotation of the trays that takes place once in three days, the barcodes of the pots are manually recorded into a database. Because these records are manually entered, there is a possibility of human error. As a future enhancement, this project focuses on getting a guaranteed optimum result. This is ensured by taking the information obtained from the color codes and the information that is manually recorded in the database and validating these against each other.

Chapter 7 - References

[1] http://fiji.sc/wiki/index.php/Documentation

[2] Burger, W., Burge, M.J. 2008. Digital Image Processing: An Algorithmic Introduction Using Java. Springer Science+Business Media, LLC, New York, NY, USA.