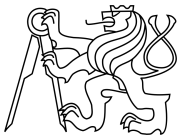
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF BACHELOR'S THESIS

Title:	Sorghum Yield Prediction Calculator
Student:	Robert Šebek
Supervisor:	Ing. Marek Žehra
Study Programme:	Informatics
Study Branch:	Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2017/18

Instructions

Design and implement an application for iOS that will allow users to estimate the yield of Sorghum farms, using image analysis and using formulas based on the research of Kansas State University.

1. Design and implement an algorithm to perform image analysis on sorghum heads, the algorithm will use the OpenCV library.
2. Design and implement an iOS application that will behave as an interface for the algorithm. The application allows farmers to upload input data and preview the output of image analysis.
3. Document and test the application and the algorithm.

The work is based on the research conducted at Kansas State University. The student was introduced to the topic during his exchange stay at Kansas State University.

References

Will be provided by the supervisor.

L.S.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague October 6, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Bachelor's thesis

Sorghum Yield Prediction Calculator

Róbert Šebek

Supervisor: Marek Žehra

5th January 2017

Acknowledgements

I would like to thank the lead researcher on the project Dr. Ignacio Ciampitti, who introduced me to the project and also guided me under the supervision of his assistant Guillermo Balboa.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 5th January 2017 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Šebek, Róbert. *Sorghum Yield Prediction Calculator*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Během výměnného pobytu na Kansas State University, jsem měl příležitost spolupracovat s vědci z fakulty Agronomie, kteří mě seznámili se svým výzkumem zaměřeným na rostlinu čirok. Pro odhad sklizně této plodiny měli vyvinutý algoritmus, který ovšem pro širší využití v praxi potřebovali převést do mobilní aplikace systému iOS. Požádali mne tudíž o zhotovení takové aplikace. Celý proces vývoje této aplikace je stručně popsán v předkládané bakalářské práci.

V první části práce jsem vyvinul algoritmus obrazové analýzy, který je schopen zpracovat obrázky hlav čiroku a odhadnout plochu rostlin. V druhé části je popsán vývoj aplikace iOS, která slouží pro zadání ostatních relevantních informací uživatelem a výpočet odhadu sklizně. Aplikace také integruje obrazovou analýzu z první části práce.

S pomocí knihovny OpenCV jsem v jazyce C++ vyvinul algoritmus pro obrazovou analýzu, jenž jsem následně prověřil na testovacích datech, která mi poskytli vědci z Kansas State University. Pro implementaci aplikace iOS jsem použil programovací jazyk Objective-C v vývojářském nástroji XCode 7. Následně jsem algoritmus obrazové analýzy integroval do již zmíněné aplikace iOS, tak aby bylo možné provést analýzu pořízením snímků hlav rostlin čiroku přímo z fotoaparátu telefonu.

Finální verze obrazové analýzy byla prověřena na testovacím balíčku 1400 obrázků. Průměrná odchylka výsledků získaných pomocí algoritmu oproti skutečným hodnotám z testovacího balíčku činí 4.825 %. Aplikace IOS umožnuje spouštět obrazovou analýzu přímo na mobilních zařízeních a výsledky analýzy kombinuje s ostatními zadanými údaji, přičemž výsledkem je zobrazení odhadu sklizně rostliny. Kromě toho také umožňuje odesílání výsledků (včetně dat, která do aplikace vložili uživatelé) na webové uložiště Firebase, kde k nim mají vědci přístup. Aplikace byla odeslána k dalšímu testování u zadavatele pomocí platformy TestFlight.

Klíčová slova Čirok, odhad sklizně, obrazová analýza, iOS, firebase

Abstract

While studying at Kansas State University as part of a student exchange programme I received the opportunity to take part on an interesting project combining the fields of Software Engineering and Agronomy. The researchers at the Agronomy department were working on a yield forecasting method for the plant sorghum and it was their vision to implement this method into an iOS application. The process of development for this application is summarized within this bachelor thesis.

The task itself can be divided into two major components. In the first part I developed an image analysis algorithm that analyses images of sorghum heads and outputs their surface area. The second part was to develop an iOS application that retrieves other relevant data from the user and integrates the aforementioned image analysis algorithm. Using the retrieved information, it performs the yield forecasting method and displays an estimated yield.

I developed an image analysis algorithm using OpenCV (open source computer vision library) in C++ that I tested on a dataset supplied to me by the researchers at Kansas State University. To implement the iOS application, I used Objective-C and the development tool was XCode 7. Finally, I integrated the image analysis component into the application and modified the C++ base code to work with the rest of the Objective-C code in the application. This allows for the image analysis to be performed on images of sorghum heads taken by the mobile device's camera.

The final version of the image analysis algorithm was tested on a dataset of about 1400 images. The average deviation of sorghum head area calculated by the algorithm compared to the actual values taken from the dataset was 4.825%. The application successfully integrates this algorithm and allows the upload of results to cloud storage using Firebase. The application was submitted to the researchers at Kansas State University using the testing platform TestFlight.

Keywords Sorghum, yield forecasting, image analysis, iOS, firebase

Contents

List of Figures

List of Tables

Introduction

The use of mobile devices has become part of the everyday life. With the introduction of the iPhone and subsequent launch of the Android platform and its many devices, we could hardly imagine our lives without mobile technology anymore. Just as smartphones have revolutionized communications, information technology has changed the landscape of agricultural practices, with what is now known as precision agriculture. We commonly hear about advancements in farming technology achieved using primarily drones and other vessels capable of applying computer vision and image analysis on areas of farming, to gain information that was previously difficult to retrieve.

Smartphones, with their ever-improving cameras and processing power, represent good candidates to at least supplement the use of these high cost devices like drones. As proof of concept, that computer vision concepts can successfully be implemented on mobile devices, we could name the rising trend of QR codes, or augmented reality.

The area this new technology could be applied to is called yield forecasting, and in this case, yield forecasting for the plant sorghum. While yield forecasting has been around since the dawn of farming, new research from Kansas State University team led by Dr. Ciampitti, shows that image analysis can be applied in this context, to improve yield forecasting of sorghum.

This bachelor thesis has the following structure:

- Since the topic of this thesis is the application of an agricultural method in information technology, I will begin by introducing the reader to the plant sorghum. I will discuss its main characteristics and the research that went into creating a model for forecasting its yield. Understanding the model and its components is crucial for understanding the software implementation.

- Further I will analyse the algorithm for analysing images of sorghum in order to get relevant data from them. I will also perform a software

analysis of the iOS application that will implement this image analysis algorithm and retrieve relevant data.

- In the implementation chapter, I will discuss the final implementation of both the image analysis and the iOS application. I will include snippets of relevant code and discuss how it works together to achieve the goal of this thesis.
- The chapter about testing, is largely devoted to the robust testing platform I built, in order to test and continously improve the image analysis algorithm. I will also describe the testing data that was provided to me by Dr.Ciampitti and his team. Lastly I will discuss the testing procedures for the iOS application.
- As a summary I will recap the successes and failures of this project, and I will discuss the improvements that could be made to both the image analysis and the iOS application.

This thesis is an extension of the research coducted by Dr. Ciampitti's team, who analysed thousands of samples of sorghum plants and documented their results. On the basis of this data they developed a forecasting method, which is the basis of this thesis.

Chapter 1

Sorghum

1.1 Foreword

This introduction to the plant sorghum serves as a bridge between the areas of agronomy and software engineering. It should help the reader understand the connection between grain sorghum yield forecasting and image analysis and how this cooperation came to exist.

1.2 Introduction to sorghum

Sorghum is a grain crop that belongs to the grass family, *Graminae* of tribe *Andropogonae*. The name comes from the Latin word "Syricum" meaning grain of Syria. Sorghum is an ancient crop and its cultivation probably originated in East Central Africa.[1]

Although sorghum is not widely known, according to FAOSTAT it ranks 31st on the world's Top 50 commodities, with over 60 million tonnes harvested in 2013 [2]. In 2005 sorghum ranked fifth in world grain production behind only wheat, rice, maize and barley, names that most people will be familiar with. [3]

Clearly sorghum is a very relevant crop on the world scale, but it holds a special significance in developing countries. Although mostly produced for animal feed in industrialized nations [1], sorghum is an important staple in the semi-arid tropics of Asia and Africa. Crops like sorghum are still the principal sources of energy, protein, vitamins and minerals for millions of the poorest people in these regions. [4]

Despite its significance and production scale sorghum is often considered a *lost* crop. *"Sorghum now receives merely a fraction of the attention it warrants and produces merely a fraction of what it could".* [5] Some researchers believe it is inadequately supported considering it is the world's fifth major crop.[5]

To give more support to these under-utilized crops, more research needs to be dedicated to them. Research needs to be focused on finding methods for

increasing yields and sustainability of these crops. A key matter to consider is however, that application of these methods needs to be both accessible and easy to use, given that nearly half of sorghum's production stems from developing countries in Africa 1.1, where education and funding are lacking. [2].

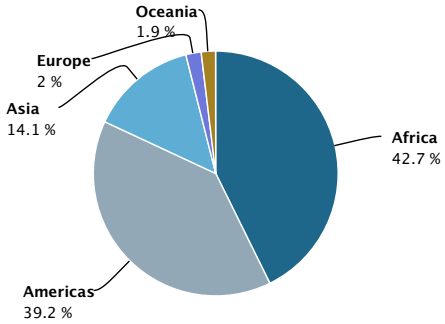


Figure 1.1: Distribution of sorghum production in 2014 [2]

With that I introduce a unique way of estimating sorghum grain yield using research conducted by researchers at Kansas State University, led by Dr. Ignacio Ciampitti.

1.3 Introduction to sorghum research

There are various areas of research in agriculture. Most people will associate crop research with developing new fertilizers or grain varieties. However, there are also areas of agricultural research, where information technology has revolutionized farming practices. The use and application of information technologies in agriculture is nowadays most commonly known as precision agriculture [6].

A key component of farming practices is crop forecasting. In 1956 an article provided by Australian Agricultural and Resource Economics Society in its journal Review of Marketing and Agricultural Economics, the author G. R. Spinks defined crop forecasting as

"A statement of the most likely magnitude of yield or production of a crop." [7] In other words, crop forecasting lets us estimate the total yield of a crop in advance of actual harvest.

Crop forecasting is of great importance in not only industrialized countries, but primarily in developing countries. In industrialized countries crop forecasting allows farmers to plan their marketing strategies and provides a basis for making decisions about crop management practices. However, crop forecasting is of greatest importance in developing countries, where it can be used as part of early-warning systems regarding food shortages. Early warn-

ing of poor harvest could allow policy makers the time to ensure food security in vulnerable areas.[8]

1.4 Sorghum crop forecasting

Having already established the importance of sorghum in developing countries, the next step is to develop a viable forecasting method for it, that can be utilized even in these underdeveloped regions. As previously stated, one such method was devised by the researchers at Kansas State University.

Before we get the procedure of estimating sorghum yield, we need to understand the main components that go into getting a sorghum yield forecast:

- Number of plants
- Total number of seeds per head
- Seeds per pound

[9]

The first component, namely number of plants, is known to the farmers well before the end of the season. The number of plants is the key components to forecasting, as a high number of plants obviously translates to a high yield. However, when planting there is an equilibrium to reach as increasing the number of plants also increases competition for resources, which can diminish the production of individual plants. [9]

While the number of heads (i.e. number of plants) is known quite early, the number of seeds per head can only be determined approximately two to three weeks after flowering. The final component, seeds per pound i.e. weight of a seed, can only be determined close to the end of the season. [9]

Knowing all the yield components that are needed for forecasting the research further suggests methods for getting a good estimation for each of them.

A key thing to consider when sampling the field is that we need to account for field variability. The research recommends to perform yield estimations in at least 5 to 10 sections of the field to account for this. [9]

1.4.1 Step 1: Number of heads per unit area

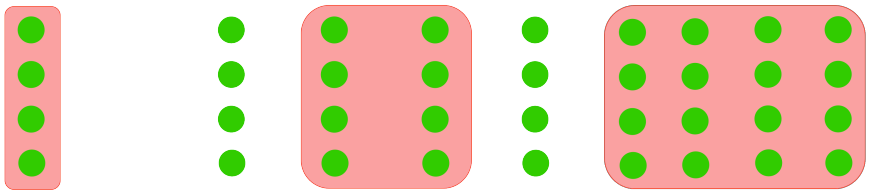
Since the research was conducted by a US based institutions, most of the following units are imperial. Therefore, the unit of area used is acre.

To be able to get the number of heads per acre we need to consider the row spacing that is used. To get an idea of what row spacing is, imagine a typical crop field. You will see crops planted in rows. The distance between these rows is referred to as ”row spacing”. The three most common ones are

30, 15 and 7.5. Since these clearly influence the number of heads per acre, we need to make adjustments for it in our formula.[9]

Since counting all plants in an acre is clearly a strenuous task, the research suggests counting heads in a fraction of the field and then multiplying by a factor, which depends on the row spacing. In this case, it is recommended to count heads in a 1000th of an acre, which corresponds to roughly 17.5 feet of row in a 30-inch row spacing. [9]

Since a 15-inch row spacing doubles the number of plants in an acre, we can either multiply the number of counted heads by 2 or just count plants in two neighboring rows, which gives us a little bit more precision. Accordingly, a 7.5-inch row spacing quadruples the number of heads, therefore we count in 4 rows. [9]



(i) 30 Inch (ii) 15 Inch (iii) 7.5 Inch

Figure 1.2: Figure representing heads that need to be counted as red.

So to get a number of heads per acre a simple equation can be used:

$$headsPerAcre = headsCounted \times 1000$$

1.4.2 Step 2: Estimation of the number of seeds per head

The seed number is the most complicated yield component to get a good estimation for. Depending on crop condition the total number of seeds can range anywhere between 100 to 5000 seeds per head. This makes counting seeds for various samples very tedious work.[9]

Unfortunately the research suggests that the number of seeds per head is the leading component in influencing the yield forecast, as can be seen in figure 1.3.

Although there are commercial grain counters available, these can cost up to a couple thousand dollars. Therefore, the researchers sought to find a simple, accessible and reliable way of estimating the number of seeds per head. For this two approaches were considered: [10]

- **Allometric estimations:** This method involves getting the width and height of the plant and then using a spherical volume equation to get an estimate for the number of seeds.

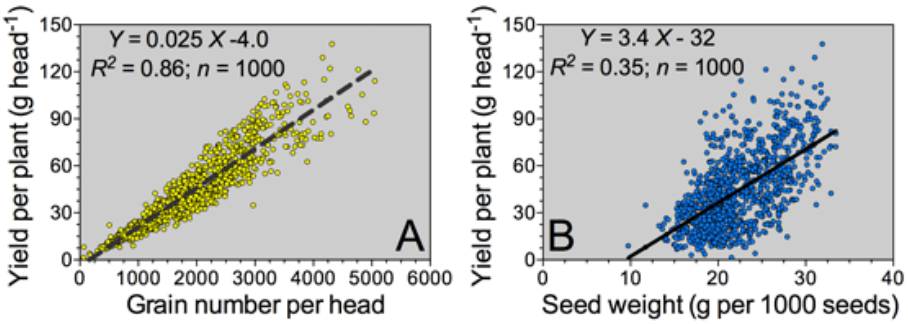


Figure 1.3: Relationship between yield components and final yield per plant.[9]

- **Head imagery:** This method suggests taking a photograph of a sorghum head sample and running in through image analysis to estimate the head volume[10]. Predictably this approach is the basis of the algorithm used for the implementation of the iOS app in this thesis.

The relation between sorghum grain count and head area is:

$$numberOfGrains = 113.6 * sorghumHeadArea + 236.38$$

The specific approach that the researchers used will be discussed in the next chapter.

1.4.3 Step 3: Estimation of the seed weight

Unlike in seeds per head, research suggests that there is only a very small variation in seed weight. The best crop averaged 25.5 grams per thousand seeds, while the worst crop scored 24.5 g/1000, which represents only a 4% difference. The difference that seed weight makes is therefore negligible. [9]

1.4.4 Step 4: Calculating the yield forecast from individual yield components

Once all critical components have been established the research gives an equation to be used for calculating a final yield per unit area, in the case acres: [9]

$$PoundsPerAcre = \frac{(Heads \times SeedsPerHead) \times 1,000}{SeedsPerPound}$$

1.5 Method for Estimating the number of seeds per head

To establish a relationship between head imagery and yields the researchers scanned thousands of sorghum heads taken on measure sheets like in figure 1.4. To perform their analysis, they used a white sheet of paper with a blue square as scale reference. The dimensions of the square are 1 by 1 inch.

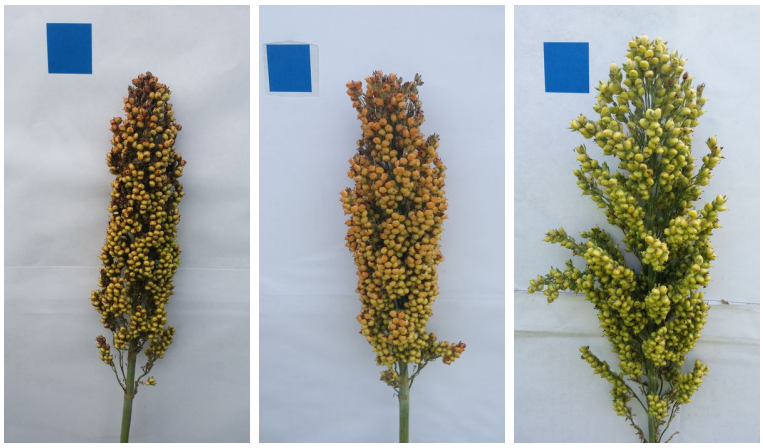


Figure 1.4: Examples of sorghum heads that were used for research, taken on a measure sheet with a blue square.

Each photograph taken was then run through a computer algorithm that extracted the plant and the square from the background and then compared their areas to get the area of the sorghum plant.

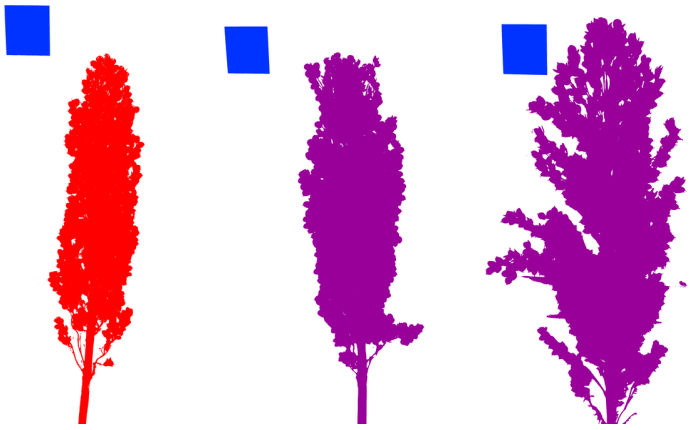


Figure 1.5: Processed images

To develop a correlation the researchers used close to 2000 images from roughly 200 different sorghum fields. To appropriately describe each field, 10

samples were taken from each field. The results of each image analysis along with observed yield for each field were carefully catalogued in a spreadsheet. From there a correlation was developed. This file along with all original photos was supplied to me for developing and testing a mobile algorithm.

Chapter 2

Analysis of image recognition

The image analysis is a cornerstone of the iOS application, because without an accurate estimation of sorghum head area, the forecasting method cannot produce a yield estimate.

2.1 Technology

While still at Kansas State University I consulted an associate professor at Kansas State University Dr. William H. Hsu. Dr. Hsu specializes in pattern recognition and teaches a Machine learning and pattern recognition course at Kansas State University. It was his recommendation to use OpenCV, an open source computer vision library. It was recommended I avoid the newest 3.X version for now and go with the more stable 2.4 version.

Other open source libraries that I considered are:

- SimpleCV [11]: SimpleCV is an open source framework for building computer vision applications. It combines several computer vision libraries, such as OpenCV and strives to make their application easier for the developer. Since it is written in python, which iOS does not have native support for, it is not the best option for an iOS app.

- ccv [12]: ccv is also an open source vision library that is application driven and runs on the iPhone and iPad. However, it is a relatively new framework and the project only began in 2010. Therefore, it lacks the documentation and user base feedback that a more established framework like OpenCV has.

I did not consider any of the closed source, paid solutions such as Matrox Imaging Library, because this is an open source project.

2.1.1 Introduction to OpenCV

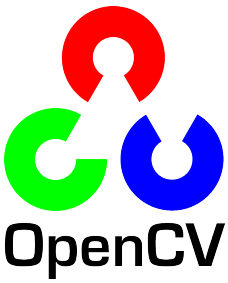


Figure 2.1: OpenCV Logo [13]

OpenCV is an open source computer vision library. Computer vision can be understood as the transformation of image data into either a decision or a new representation. This is usually done with the purpose of achieving a goal such as monitoring changes in image feed for motion detection.[14]

The library is written in C and C++ which makes it favourable for use in iOS development as the compiler allows for a mixture of Objective-C and C++, known as Objective-C++[15]. This makes integration of an image analysis algorithm into an iOS application effortless. For this reason, I also decided to use Objective-C in favour of the newly emerged and very popular Swift programming language.

OpenCV was designed for computational efficiency and can take advantage of multiple processors.[14] This is important as having an image analysis process running on an iOS device can have a high complexity and optimization needs to be a top priority.

Although OpenCV may seem like a humble open source project, it dates all the way back to 1999 and has found its uses in many top companies such as IBM, Microsoft, Intel, SONY, Siemens, and Google.

2.2 Approach

Image analysis of sorghum heads can be split into these two parts.

- Identifying the **blue square** and finding its area as percentage of the image
- Identifying the **sorghum head** square and finding its area as percentage of the image
- Calculating the physical world area of the sorghum head as a ratio of the previous values

2.2.1 Square detection

Since basic shape recognition is something that computer vision algorithms have done for decades now, I started off by researching current solutions. Because I am using the OpenCV library, I began searching for solutions that are based on this library.

OpenCV has a working square detection sample algorithm in C++ hosted on their github available under the reference in source [16]. The program performs the following steps to find all squares on the image:

In the first part, it retrieves all contours from the image. Here is the part of the code responsible for it. Note that to shorten the amount of code pasted in this thesis, I shortened the code, by removing commented parts:

```
1     pyrDown(image, pyr, Size(image.cols/2, image.rows/2));
2     pyrUp(pyr, timg, image.size());
3     vector<vector<Point> > contours;
4     for( int c = 0; c < 3; c++ )
5     {
6         int ch[] = {c, 0};
7         mixChannels(&timg, 1, &gray0, 1, ch, 1);
8         for( int l = 0; l < N; l++ )
9         {
10            if( l == 0 )
11            {
12                Canny(gray0, gray, 0, thresh, 5);
13                dilate(gray, gray, Mat(), Point(-1,-1));
14            }
15            else
16            {
17                gray = gray0 >= (l+1)*255/N;
18            }
19            findContours(gray, contours, RETR_LIST,
                  CHAIN_APPROX_SIMPLE);
20            vector<Point> approx;
```

Listing 2.1: [16]

[16]

This part of the code performs the following procedures:

- Scales the image down to half size and then up to the original size again in order to remove noise. This is achieved using the *pyrDown* and *pyrUp* functions that downscale and upscale images respectively. These functions also apply a blur. [17]

- Iterates over all three color channels. Switching of the channels is achieved using the *mixChannels* which simply copies the desired colour channel from original array *timg* to the destination array's *gray0* 0 index. [18]

- Iterates over N thresholds and also attempts Canny thresholding, creating a binary image.

- Uses *findContours()* to find all contours in the binary image "gray" and saves them to a vector of contours *contours*. The last parameter

CV_CHAIN_APPROX_SIMPLE is especially important here. This parameter sets the mode for *findCountours()* and tells it to compress horizontal vertical and diagonal segments and leave only their endpoints. That means that upright rectangular contours will be encoded with only 4 points. [19]

Once contours have been retrieved the algorithm must iterate over the vector of contours and decide which ones are squares. The following part of the code is responsible for that:

```
for( size_t i = 0; i < contours.size(); i++ )
{
    approxPolyDP(Mat(contours[i]), approx, arcLength(Mat(contours
        [i]), true)*0.02, true);
    if( approx.size() == 4 &&
        fabs(contourArea(Mat(approx))) > 1000 &&
        isContourConvex(Mat(approx)) )
    {
        double maxCosine = 0;
        for( int j = 2; j < 5; j++ )
        {
            // find the maximum cosine of the angle between joint
                edges
            double cosine = fabs(angle(approx[j%4], approx[j-2],
                approx[j-1]));
            maxCosine = MAX(maxCosine, cosine);
        }
        if( maxCosine < 0.3 )
            squares.push_back(approx);
    }
```

Listing 2.2: [16]

In this code snippet, the algorithm performs the following:

- Iterates through all the contours in the vector.
- Approximates a polygon from a contour and saves the approximation to the variable *approx*. For this the *approxPolyDP* function is used. The function takes as input the contour from the vector and attempts to approximate it using an approximation accuracy parameter, specified in the third parameter. This accuracy parameter sets the maximum allowed distance between the contour and its approximation. The last parameter specifies whether the contour is closed. If true the approximation is closed, meaning that the first and last vertex are the same.[19]
- Checks if the approximated contour has 4 vertices, is at least 1000 pixels in size to filter out noisy contours and checks whether the polygon is convex. If all these requirements are met, we still need to perform a check for the angle degrees, to make sure we filter out all non-square 4 sided polygons, such as parallelograms or kites.

- Calculates the cosines of all tuples of the accepted polygon and stores the largest one in the variable *maxCosine*. Since all angles within a square need to be the same, it is satisfactory to only consider the largest deviation from the 0-cosine value all the angles should have. We can then specify a desired accuracy value to filter out imperfect squares.

This algorithm serves as a very useful basis for the algorithm we will be using. Under these conditions however, the algorithm picks up rectangles as well, because they also have 4 vertices, are convex and have roughly 90 degree angles. Also, this algorithm searches for squares of unspecified size and colour and even searches for gradient squares. The algorithm complexity here is quite high, because it must perform the following number of operations:

$$numberOfOperations = 3 \times (numberOfThresholds + 1) \times numberOfCountours$$

Since we know that we are looking for a square within a reasonable size interval and specified colour spectrum we can optimize the algorithm, in some ways. The proposed optimizations will be discussed in the chapter implementation.

2.2.2 Sorghum head detection

Unfortunately, there is no sample algorithm for detecting sorghum heads, as is the case in detecting squares. Detecting sorghum heads is tricky, because as mentioned previously, sorghum comes in many different colours, sizes and shapes and these variations need to be accounted for. Fortunately, we can rely on the fact, that sorghum has a high saturation and it is therefore easy to threshold it from the white paper background if we perform analysis on an HSV formatted image. HSV stands for "Hue-Saturation-Value", where the saturation component can be extracted. Knowing that the white paper background will have a saturation value close to 0 can help us make a thresholding prediction.

Here is a basic rundown of what an algorithm detecting sorghum head area might look like

- As standard in computer vision, we attempt to reduce noise in the image and prepare the image for analysis.

- In order to find contours in the image, we need to create a binary representation of the image. Thresholding is unsatisfactory in this case, because sorghum stems might have a different colour saturation from the sorghum grain, resulting in a colour gradient that would not threshold properly. Therefore, we will use canny edge algorithm here, which can detect edges even in gradient objects.

- Canny edge detection leaves us with a scattered array of white pixels, which represent found edges. These pixels might not be adjacent and therefore might not form a contour around the plant. To connect these scattered pixels, we will use dilation, which simply expands these white pixels using a specified setting, which should form a connected contour around the sorghum head.
- As in the squares algorithm, we will need to use the *findContours* method.
- Since we know nothing about the shape of the sorghum, we cannot make any assumptions about the shape of the contour, as we did in the squares algorithm. Therefore, the only thing to rely on, is that the sorghum head will likely be significantly larger than the square and since it is the only object in the image, its contour should have the largest area. Therefore we simply iterate through all the contours to find the largest contour and store it.

2.2.3 Calculating sorghum head area

Once we have the area of the blue square area as percentage of the image $squareAreaPercantage$ and the sorghum head area as percentage of the image $sorghumHeadAreaPercentage$. Knowing that the area of the square is exactly 1 $inch^2$ we get the following equation.

$$sorghumHeadAreaInches = \frac{sorghumHeadAreaPercentage}{squareAreaPercantage}$$

2.3 Knowledge gained by analysis

Having analysed both algorithm parts, namely getting the area of the square and getting the area of the sorghum head a pattern emerges. We can see that both parts of this algorithm perform similar procedures and therefore it would probably be possible to unite these two algorithms and only perform some steps once. Image analysis preparation, converting to binary image and finding contours occur in both parts of the algorithm and can therefore be possibly performed only once and their results used for both parts of the algorithm. We will refer to this optimization in the implementation section.

Chapter 3

Analysis iOS application

Mobile technologies have found their application in various areas of everyday life. The coupling of mobile technologies along with computer vision, for tasks like scanning barcodes has proven to be especially useful.

For example, the retail giant Lowe's has recently deployed over 42 000 iPhones in retail overhaul to serve as point-of-sale systems. [20] Applications like these illustrate that the iPhone is not merely a consumer device anymore and can be used for a variety of critical tasks.

In this thesis, I will attempt to use an iOS device to gather information about a sorghum plant, by analysing images of sorghum heads. Furthermore, the application should analyse this data to provide a yield estimate and then store the data along with the estimate online.

3.1 Technology

The first thing to consider is the choice of development tool for iOS development. While Xcode is, the Apple supplied IDE, there are still other options to consider, such as cross-platform tools like PhoneGap and Xamarin. There are two major reasons why I decided against using a cross-platform solution for the implementation of the app.

- According to Developer Economics research CPT (Cross platform tools) are only used by 29% of developers for iOS development. Since this is a student project and one that is likely to be passed on to other students after my graduation, it makes most sense to choose a development platform that most people will be familiar with. Since native development market share is for now still overwhelming CPT, I saw it as advantageous to use native development, especially if we consider that the 29% CPT developers are likely scattered across a plethora of different tools. [21]

- Many online sources report that these hybrid solutions suffer in efficiency and run a lot slower than native counterparts. Therefore, they are generally not recommended for high-performance apps.[22]

3.1.1 Xcode

For abovementioned reasons Xcode 7 was chosen as the development tool. Most of the development was done in Xcode 7 in Objective-C. To make the app compatible with iOS 10 some minor bugs had to be fixed in Xcode 8.

3.2 App requirements gathering

The basic requirements for the application were gathered in several meetings with Dr. Ciampitti and his staff. The basic breakdown of the required features the application must implement can be summarized with the following points:

- Entering required field data
- Creating a visual tutorial to guide user through the process of selecting / capturing images of sorghum
- Interface for uploading, processing images and previewing image analysis results
- Interface for displaying yield results
- Final screen with useful information
- Backend for storing data, results, and images

This breakdown can be understood chronologically, that means that the requirements are listed in the order that they appear in the application. We need to collect data first to make calculations after images are uploaded, and we need to show a tutorial on image uploads, before the user gets to the upload interface.

3.2.1 Field data

3.2.1.1 Collecting required field data

To calculate the resulting yield forecast we need to establish the following components:

- Field name
- Number of acres
- Row spacing

- Plants per 1/1000 acre
- Seeds per pound[1]

Gathering this data will require user input in form of various input field types, ranging from basic text fields to radio buttons. The specific implementation of collecting these components will be discussed in the next chapter.

3.2.1.2 Collecting location information for report

Although the location of the field is not accounted for in this crop forecasting method, it is a crucial piece of data for further research. Knowing where the crop forecast was made would provide invaluable information to the researchers, as they could use it to improve the forecasting method and create powerful prediction models, based on location.

It was therefore one of the top priorities of this app, that whenever a user makes a forecast and is connected to the internet, at least some form of location data should to be included. Since iOS, by default requires permission to gather GPS data, if the user refused to allow location data services there would be no location data available.

Therefore, I proposed a system, which would let the user decide whether he wants to consent to location data collecting, and depending on the answer either:

- Collect data automatically using built in iOS methods
- Ask user to pick his location using a preselected list of locations

Since it would be difficult to store an offline list of all possible locations in the world, it was agreed upon, that the app would provide basic country data location for all countries besides the USA, meaning the user would only have to select a country. In the USA, however a greater precision was needed all the way down to county levels. The specific way this data was gathered and presented to the user for selection is specified in the implementation part of this thesis.

3.2.2 Storing field data

To facilitate the storage of entered field data, a data structure was needed to store and hold field data. While the requirements didn't require any persistent storage of results, I decided the use of persistent storage would allow better upgrade options, just in case the app needed to be expanded in the future. For example, if it was decided that the users should be able to view the results of past yield forecasts they performed.

[1]Seeds per pound is only required at the end of the process, but it makes sense to group it with the other direct input components

There are various solutions for persistent storage on the iOS platform. We can group iOS persistent storage alternatives into these three categories:

- **NSUserDefaults**[23]: NSUserDefaults class primarily provides a programmatic interface for interacting with the defaults system, which allows an application to customize its behaviour to match user preferences. These preferences are stored as a key-value dictionary that can store basic datatypes. It is a simple solution for storing small amounts of data. While this is a simple solution for implementation it carries a major disadvantage, namely that it does not support data querying.[24]
- **Core Data**[25]: Core Data is a framework that manages the model layer objects in iOS applications. It provides generalized solutions to object life cycle management, including persistence. The downside of Core Data is that it is a very complex framework that is also quite outdated in competition with modern frameworks like Realm [24]
- 3rd party solutions like **Realm**[25][26]: Realm is an offline-first mobile database that runs directly inside phones, tables or wearables. It offers Objective-C support and promises to be faster than its alternatives SQLite and Core Data.

Upon weighing these options, I decided to go with Core Data since it is a native iOS solution and it allows designing of the data model right inside of Xcode.

3.2.2.1 Data model

The data model pictured in figure 3.1 is used to hold field data components as they are collected throughout the application. It was modelled in Core Data in Xcode 7.

The central object is *FieldMeasurement*, which encompasses all the related field data such as *numOfAcres*. It has a "to-one" relationship with both *ManualGPS* and *AutoGPS*. This facilitates the storage of location data depending on whether the user wants to share his location using the built-in GPS sensor, or wants to manually select his location from a database of locations. We can see a "to-many" relationship with *Measurement*. We can think of *Measurement* as an object that stores one image analysis. Therefore, it stores the resulting *appArea*, *measurementID* and most importantly *processedImage*, which is a binary data representation of the processed image.

3.2.3 Creating a visual tutorial to guide user through the process

As with any user oriented application, it is only reliable if the user knows how to use it. In the case of image analysis, taking the right photos of the

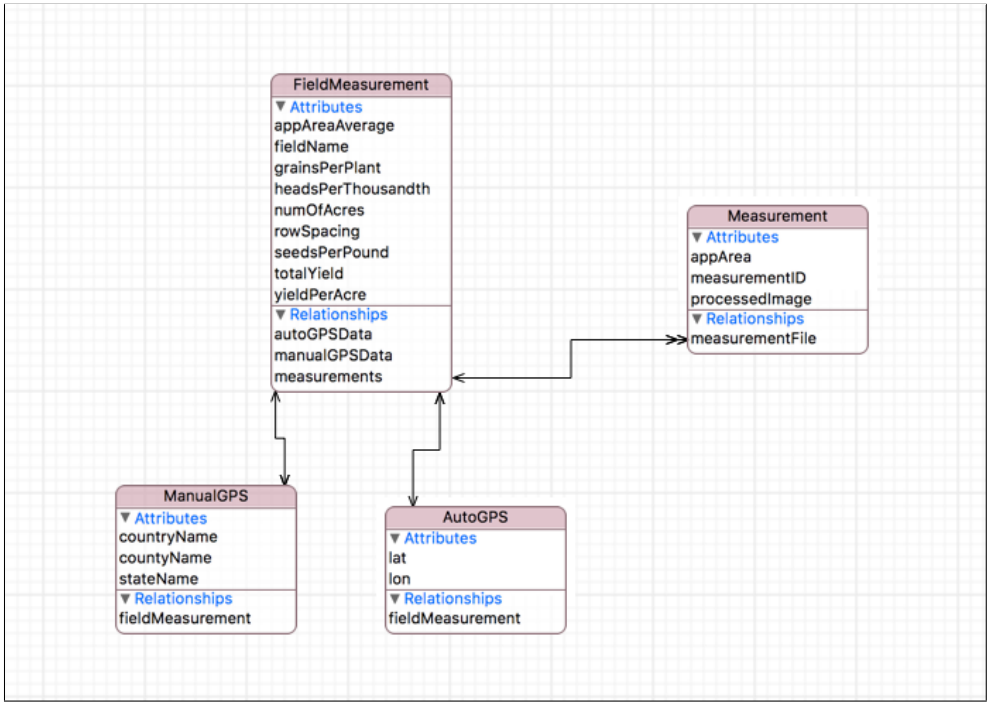


Figure 3.1: Data model

plant is crucial to getting a good estimate and a correct reading. Although the algorithm does account for some user error, and discards invalid images, it cannot account for all unpredictable photo compositions that the user might think of creating, without proper guidance.

The tutorial needs to explain the following steps to the user:

- Preparing or downloading and printing a measure sheet
- Placing the plant on the measure sheet correctly
- Understanding what a valid result should look like

Furthermore, the tutorial should help the user avoid some common mistakes such as having the plant outside the bounds of the measure sheet or having the plant touch the square. Both mistakes can cause the algorithm to fail or in worst case scenario display inaccurate results.

Algorithm failure is not a major cause for concern, as the results of a failure of the analysis is recognized by the algorithm and discarded. However, inaccurate results are a twofold problem. Firstly, they can cause the result of the forecasting to be significantly beyond what a reasonable range of yield would be and therefore cause the user to doubt the underlying algorithm. Secondly, if the results of an inaccurate analysis go through the application

unnoticed, they could be submitted to the researchers as valid research data and cause inconsistencies in the database.

To increase app reliability a secondary check needs to be implemented. If an inaccurate reading passes by all the algorithm checks, the only way to recognize the inaccuracy is to have the user verify the result personally. This system will be discussed in the following sections.

A tertiary data validation stage to increase data reliability on the database level is that image analysis results will be stored on the backend and researchers will be able to analyse whether the algorithm made a correct reading. The database needs to support the deletion of invalid results by the researchers.

3.2.4 Designing user interface for uploading, processing images and previewing images

This part of the user interface is responsible for getting the images for analysis from the user's device. The user has two options of supplying these images. The user can either upload these images to the app from the device gallery or bring up the camera screen and take a photo. After every upload/photo taken, a secondary screen appears that previews the result of the image analysis.

Any image that is selected from the phone's gallery is checked against images already uploaded for duplicity and is only analysed if it hasn't been selected yet. Otherwise the user gets an error message.

Upon selecting or capturing an image the image analysis is performed and in case of a successful reading it is displayed in the review screen. This screen previews the image analysis result to the user and allows him to decide whether the image analysis was successful. The correct way to analyse the results will be explained to the user in the visual tutorial discussed previously.

In case this secondary check is successful the image is displayed in a table of valid readings and stored for the final calculation. Figure 3.2 illustrates the entire process of handling provided images

Once a specified number of images has been provided and successfully analysed. The user is prompted to advance to the next section, which shows them a result of their yield forecast.

3.2.5 Interface for displaying yield results

At this point, we have enough information to perform all calculations based on the yield forecasting method. Therefore, this interface is simply responsible for displaying the results of the yield forecasting process. While the main consideration is displaying the final yield prediction in bushels/acre, it would be helpful if the user saw a breakdown of how the calculation came to exist and saw all results of necessary sub calculations.

The only problem at this stage is that we still have not established a component that goes into the calculation, namely *Seeds per pound*. As discussed in

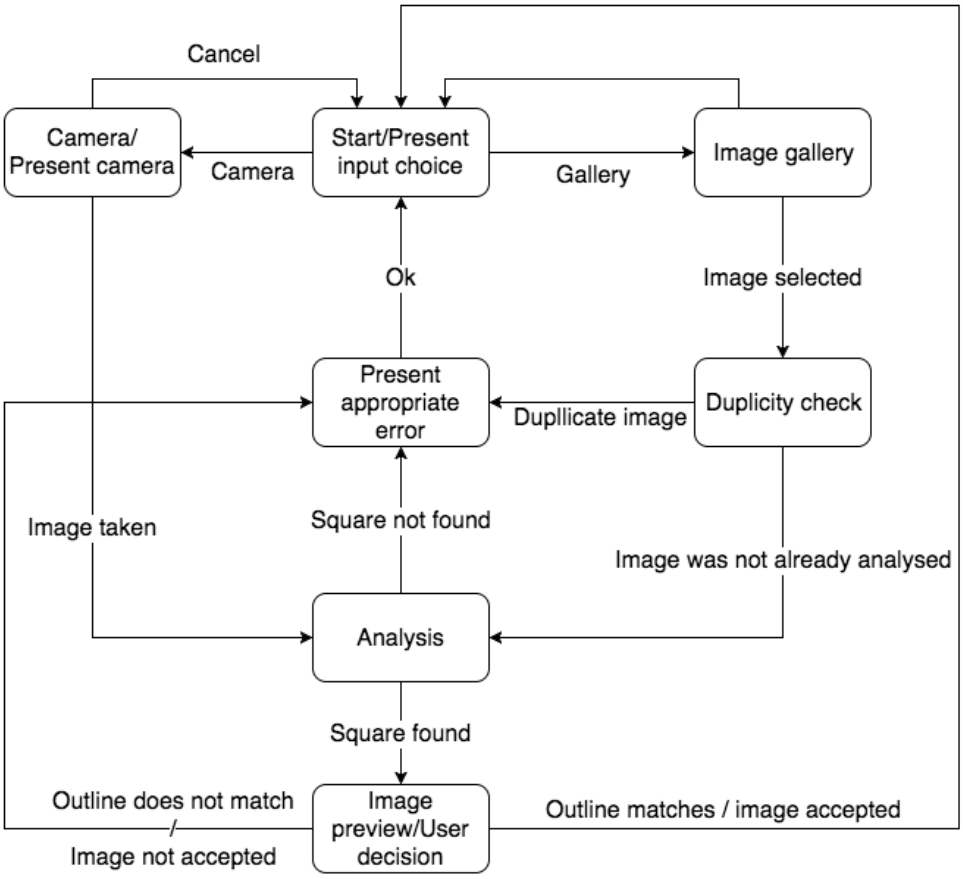


Figure 3.2: Image input and analysis process

introduction to sorghum, seeds per pound is a component that is very difficult for farmers to establish and since it does not influence the resulting yield in a major way, we can simply show farmers a range that the resulting yield could be in.

However, since the farmers might know how many seeds per pound they have, we should also give them a way of entering it and getting a precise result.

3.2.6 Final screen with useful information

Besides displaying the final yield number again, this screen is responsible for displaying some additional information that might be helpful to the farmer.

Therefore, the final screen should also feature links to relevant research. Besides that, the final screen will contain a text message thanking the user for using the app, and giving them instructions on how to report their resulting yields to the researchers, after harvest is done.

3.3 Design backend to store measurement data and images

All the application functionality is performed offline so it does not require extensive database communication. In fact, the application only needs to perform outgoing communication, because it only facilitates the sending of reports. Another aspect to consider is that the backend is for researchers and they need a way of easily navigating the data they receive. Therefore, I decided to use a backend service that provides out of the box functionality and has a graphical user interface for viewing user data. I decided that Firebase best matches these requirements.

3.3.1 Firebase

Figure 3.3: Firebase logo [27]

Firebase [28] is a cloud service that provides both a real-time database and cloud storage, which is ideal for both our research data and the images we need to store. Since the database is NoSQL the implementation of database communication will be quite effortless, as we can simply push a JSON representation of the collected data to the server, without setting up a backend SQL structure. JSON data will be stored under a unique timestamped ID, so the data can be sorted chronologically.

The images will be saved in the cloud storage, under a folder with the data timestamped ID, so we can reference them to the data. Only analysed images will be saved and they should be compressed to a reasonable size. We need to keep in mind that sorghum fields can be in very remote areas, where internet connectivity might be very limited, therefore data efficiency is a top priority. The images stored in the database are only there so that the researchers could inspect the images to make sure the analysis was successful and precise. Therefore, a lower quality of images is not an issue.

The biggest advantage of Firebase is that it provides a user interface for navigating stored data. That provides an easy access to the data for researchers.

Chapter 4

Image analysis implementation

As mentioned in the analysis of the image analysis component, there are two parts to a getting a successful reading of sorghum head area. We must first establish the area of the square and use that as a reference point to calculate a real-world area from the sorghum head area. Since both the algorithm for getting the square area and the algorithm for getting the head area share common procedures we will execute these first and then perform the unique parts of each analysis afterwards.

Since this already pertains to implementation, the code snippets posted here will be directly from the app and not from the testing program that I developed separately in C++ to continuously test and improve the algorithm. Therefore, the language is the mixture of C++ and Objective-C called Objective C++

4.1 Common steps

```
1 Mat src = [self cvMatFromUIImage:image];
2 vector<vector<cv::Point> > contours;
3 Mat src_gray =Prep(src);
4 int imageSize = src.rows * src.cols;
5 Mat canny_output;
6 Canny( src_gray, canny_output, thresh, thresh*2, 3 );
7 dilation(canny_output);
8 findContours( canny_output, contours, hierarchy, CV_RETR_EXTERNAL
    , CV_CHAIN_APPROX_SIMPLE );
```

./tex/commonSteps.cpp

In the first step, we simply convert the iOS representation of a bitmap image called *UIImage* into the OpenCV used format called Mat. The implementation of this conversion is kindly provided by OpenCV in their iOS section available here [29].

In the next step, we must perform the image preparation, since the implementation of this plays a significant role to getting a good result I will discuss the preparation in a separate section.

Once the image is returned from the *prep()* function, we need to establish an image area to get context for sizes. This calculated area is later used to make assumptions about acceptable square sizes.

Since the image is in greyscale and represents the saturation element of the image we can perform the Canny edge algorithm to find all objects that are not paper. For that we use the OpenCV procedure *Canny()*, which implements the canny algorithm [30]. Since the threshold parameter supplied here plays a big role in the outcome of the image analysis, the implementation was tested using multiple threshold values. The process of improving algorithm is further described in the testing section 6.1.

After that the scattered edge pixels are dilated to get connected outlines of the objects. For that the *dilate()* procedure is used [17], using a rectangular structuring element of a variable size. The size of the structuring element also plays a large role in the outcome of the image analysis and therefore was also subject to further testing and continual improvement.

Lastly the *findContours()* procedure was used to find all contours in the image [19] and saved to a vector of contours for further analysis.

4.1.1 Image preparation

```
1  Mat prep(Mat & src ){
2      if(src.rows>src.cols){
3          cv::Size size(1440,1920);
4          resize(src, src, size);
5      }
6      else{
7          cv::Size size(1920,1440);
8          resize(src, src, size);
9      }
10     Mat src_gray(src);
11     Mat hsvImg;
12     cvtColor(src, hsvImg, CV_BGR2HSV);
13     Mat channel[3];
14     split(hsvImg, channel);
15     src_gray = channel[1];
16     blur( src_gray, src_gray, cv::Size(5,5) );
17     return src_gray;
18 }
```

./tex/prep.cpp

The first step performed is resizing to a specified size, because iOS cameras by default have a very varied range of photo resolutions. While the resolution of the image in reasonable ranges does not affect the image analysis significantly, it is still good to standardize the image size for image analysis, because

it affects the processing time for each image. Newest iOS devices have over 12 megapixels in resolution and the trend of increasing pixel count is likely to continue [31]. Per my personal testing, resolutions past 1920p do not improve the image analysis results significantly and keeping the extra pixels is therefore not worth the processing time trade-off. To resize the Mat structure I use the *resize()* procedure documented here [32].

In the next step a copy of the Mat is created using the copy constructor described here [32].

The image is then converted to a HSV representation and stored in the variable *hsvImg*. To perform the RGB-HSV conversion the *cvtColor* procedure is used as documented here [33].

The individual channels of HSV, namely Hue, Saturation, Value are then split into an array of Mat structures using the *split()* procedure [18]. Through the process of trial and error, I established that the saturation component is saved into the second channel and therefore I copy this channel of the Mat array into a separate Mat of the same size. By doing that we create a saturation representation of the original image in greyscale.

In order to reduce noise a slight blur is applied using the *blur()* procedure [17].

The prepared greyscale image is then returned for further analysis.

4.2 Square detection

```
1  double mySquareFinder(vector< vector<cv::Point> >  & contours,
       vector<cv::Point> & square, int imageSize){
2      vector<vector<cv::Point> >::iterator it;
3      double lowerThreshHoldSize = 0.01;
4      double upperThreshHoldSize = 0.09;
5
6      double curBiggest = 0 ;
7      vector<cv::Point> approx;
8      for(it = contours.begin();  it != contours.end();it++ ){
9
10         approxPolyDP(Mat(*it), approx, arcLength(Mat(*it), true)
               *0.02, true);
11
12         double squareSizePercent = fabs(contourArea(Mat(approx))/
               imageSize);
13
14         if( approx.size() == 4 &&
15            squareSizePercent > lowerThreshHoldSize  &&
                  squareSizePercent < upperThreshHoldSize &&
16            isContourConvex(Mat(approx))
17            )
18         {
19             double maxCosine = 0;
20             for( int j = 2; j < 5; j++ )
21             {
```

```
22                  // find the maximum cosine of the angle between
                        joint edges
23                  double cosine = fabs(angle(approx[j%4], approx[j
                        -2], approx[j-1]));
24                  maxCosine = MAX(maxCosine, cosine);
25              }
26
27              if( maxCosine < 0.25){
28                  if(squareSizePercent> curBiggest) {
29                      curBiggest = squareSizePercent;
30                      square = approx;
31                  }
32
33                  return squareSizePercent;
34              }
35
36          }
37      }
38      return 0;
39  }
```

./tex/squaresImplement.cpp

For the implementation I slightly modified the sample algorithm that OpenCV provides, which was already discussed in the analysis chapter. I implemented the following improvements:

- Set a lower and upper threshold for the size of considered squares. If the square is smaller than 1% of the image, then it is too small to be considered. If the square is larger than 9% of the image, then it is likely a faulty recognition and therefore should not be considered.
- Since multiple contours are sometimes found around the same square, in order to get the outermost one I simply store the largest area result so far and check each new square area against it. If no square is found the area of 0 is returned, which is understood as not found.

If no square is found the image is treated as invalid and the app needs to handle informing the user about the unsuccessful analysis. To highlight the square a contour of the largest square is returned as well as the area.

4.3 Sorghum detection

As mentioned in the analysis chapter, we cannot make any assumptions about the shape of sorghum heads, because they come in a variety of shapes. Therefore, the only qualifying parameter in the implementation is that the sorghum head needs to be larger than a specified value *minPercentContour*, which is set to 5%. The implementation therefore simply iterates through the contours and finds the largest contour, which is presumably the sorghum head.

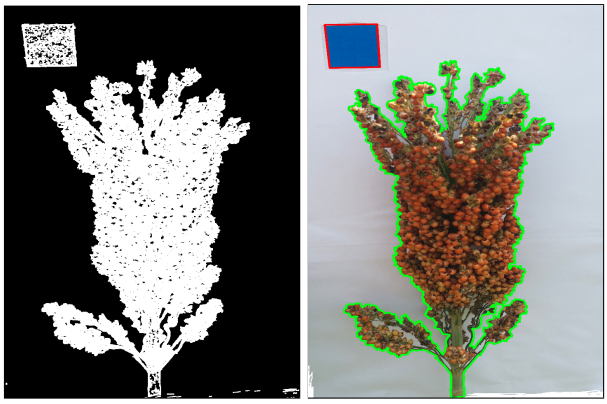
```
for(it = contours.begin();  it != contours.end();it++ ){
    float sizeLimit = imageSize/(100/minPercentContour);
    double area =contourArea(*it);
    if(area>=sizeLimit){
        if(area > largestContourArea){
            largestContourArea=area;
            largestContour = *it;
        }
    }
}
```

./tex/sorghumImplementation.cpp

4.4 Final result

Once both areas are established, both found contours are drawn onto the original image to highlight the objects found as square and sorghum head respectively. Furthermore, the real-world area in $inch^2$ of the sorghum head is calculated using the square area as a reference point using the formula from the analysis chapter. The image analysis algorithm then returns the modified image and the result as a double to the app to process it and display the appropriate user interface.

(i) Original (ii) HSV (iii) Canny edge detect



(iv) Dilation (v) Find contours

Figure 4.1: Image analysis process

Chapter 5

Application implementation

Besides incorporating the fundamental corner stone of this sorghum forecasting method, namely image analysis of sorghum heads, the app must provide supplementary functions, discussed in analysis.

5.1 Data gathering

In this section I will describe the specific ways of gathering data required for the application of the sorghum forecasting method.

To make the next section easier to understand I will discuss the individual components, that are part of the calculation, chronologically as they appear in the iOS application.

5.1.1 Field name

To improve the application's accessibility and to maintain offline capabilities, it was decided that a login/signup system would be omitted. To get a unique identifier of the field at least a string representation of a field name is required. This could make navigating the research database easier, as successive readings from one farm can be grouped and changes over time can be tracked. Since checking the validity field names would be impossible, and it is not critical for the functionality of the app or the validity of research data, no input checks will be performed on this entry.

5.1.2 Number of acres

Since this is just a numerical value that has a large variation (a field can be anywhere from 1 to several thousand acres) a simple text field will be used to retrieve this value. To maintain app reliability and ensure that no invalid data gets entered a numerical keyboard is to be used and an input check is implemented.

The input check validates that the number does not have any leading zeroes and at least a single digit has been entered. Since the number pad keyboard only allows input of digits 0-9 [34] the conversion to integer is safe if an integer size check is performed first. The specification did not specify a maximum farm size, however there are no sorghum farms larger than 10 million acres so I chose that as the upper threshold. The conversion will therefore always be safe, as the integer on even 32-bit iOS devices has a maximum value of 2147483647 [35].

5.1.3 Row spacing

Because row spacing can only be one of three different numerical values (30,15,7.5), I opted to use the native iOS picker view to handle the displaying and selecting these values. The big advantage of a picker view is that it ensures that one of the three values are chosen and therefore ensures data validity radio button is that it ensures data validity.

5.1.4 Plants per 1/1000 acre

As opposed to previous components, which are basic farmer knowledge, getting the count of plants in 1/1000 of an acre might be a new concept to some farmers. Therefore, the process of counting the sorghum heads needs to be explained to the farmer using a chart, so they understand how to obtain this data.

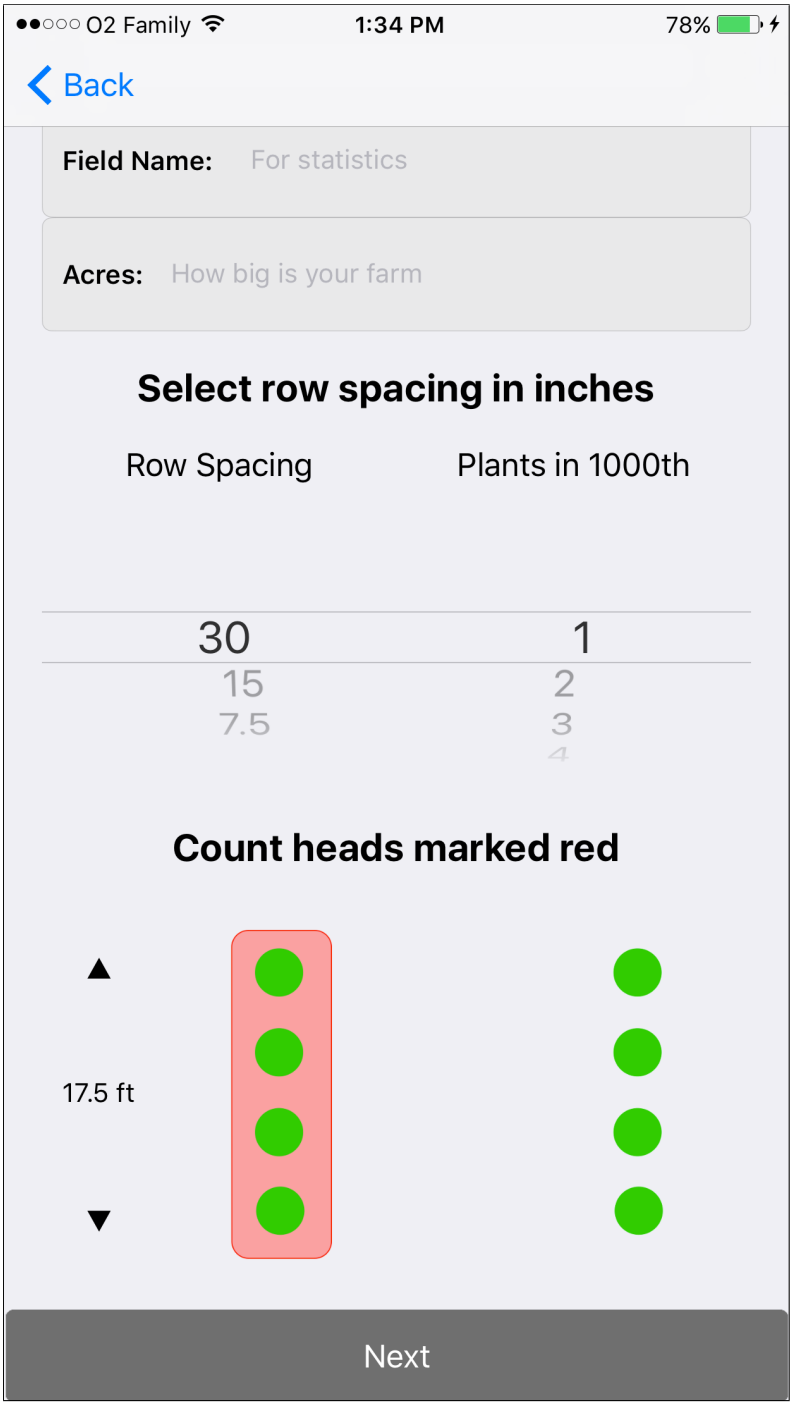
Since the counting technique is unique for each row spacing value the chart needs to update whenever a new value is selected in the picker view for row spacing. The charts displayed will be the same ones as pictured in the sorghum introducation 6.1.

Entering the value for plants per 1/1000 acre is performed using a picker view as well. The values for plants per 1/1000 acre are integers in the range 1-50 and therefore the picker view is ideal to retrieve this data.

5.1.5 Automatic location data gathering

If the user gives consent to gathering the location data automatically we can use the Core Location framework to retrieve their location. Core Location allows the programmer to specify a degree of accuracy ranging from meters to kilometres. However, higher accuracy causes Core Location to power up additional hardware and therefore waste battery power for unnecessary accuracy.[36] Since fields in the same region will have the same climate and soil conditions, getting a regional location is satisfactory. Therefore, we will settle for a kilometre range of location.

If the location service is successful and returns a location, the latitude and longitude are saved into the core data object and the next screen is shown automatically.



(i) Final design of the picker view system

If the location services fail for whatever reason the process will continue as if the user disallowed automatic location data gathering.

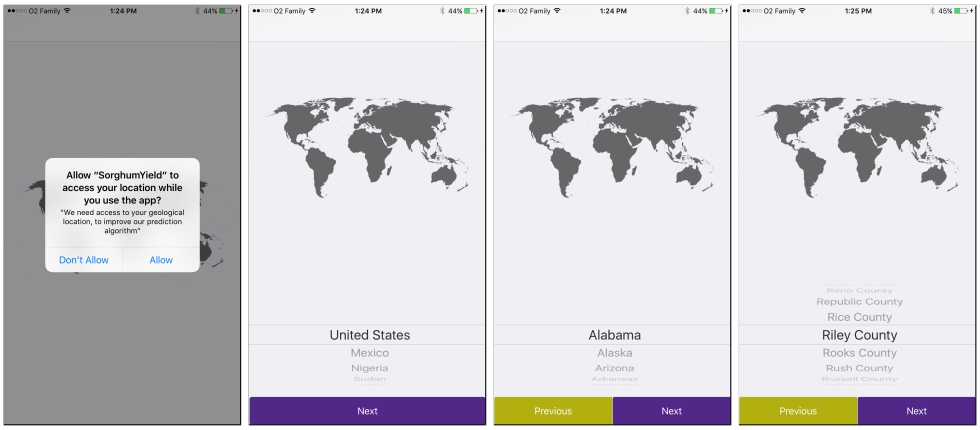
5.1.6 Manual location selection

Some users might not be comfortable with sharing the location of their farms, especially as the report submitted contains relatively sensitive information about the state of their crop, which translates to revenue. In fact, some new research suggests that almost 50% of users are concerned with privacy, while 19% see location services as a drain on batteries. [37].

Therefore, a way had to be devised to offer the user a way of putting in their location information to a degree that keeps them anonymous and doesn't drain the battery. To keep data consistent, it is better to let them choose from a list of preselect locations.

I decided to let the user select their location using a system of picker views. For that I had to gather data about countries of the world and then rank them according to their sorghum production, so that the most likely countries will be on top. Since the researchers are mostly focusing on the USA for now, the level of precision had to be adjusted in the case that the user selected the USA.

Therefore, if the user selected the USA, a second picker view is presented with a choice of states and then a third with a choice of counties of that state. Since a single county will have hundreds of farms, almost all users should be comfortable with such a degree of accuracy, as it still keeps them anonymous.



(i) Consent screen (ii) Country (iii) State (iv) County

Figure 5.2: Location picking screens

5.2 Creating a visual tutorial to guide user through the process

Following figure shows all the tutorial screens I created for the tutorial. The captions represent the text that will be displayed on screen accompanying these images.

Tutorial pages 5.3i through 5.3iv show the user how to correctly place the plant on a measure sheet. Tutorial page (5.3ii) will also have a button in app, that will download a measure sheet, which contains the square and can simply be printed on a colour printer.

On the flipside pages 5.3v through 5.3vii, show common mistakes that could be performed while taking images of sorghum heads on measure sheets. The mistakes section is separated from the first section of the tutorial by one page letting the user know that following tutorial pages showcase mistakes.

To show these tutorial pages I needed to find the most convenient user interface solution. I believe I found that solution with a so called "Page View Controller" that iOS supports natively and has a default implementation for [38]. Page view controllers simply show an image from an array of images and upon swiping left or right shows the previous or subsequent image respectively.

You can also customize the screens, which display the images with additional user interface elements to show additional information. I added a label to each screen, which displays information about the step currently shown on the tutorial image.

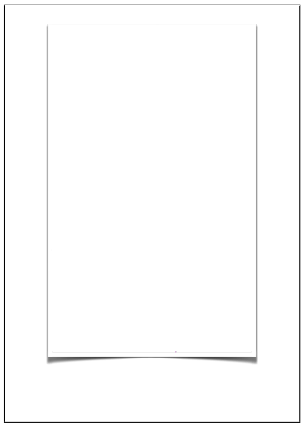
Upon swiping right from the last image, the application transitions straight into the image picking/ image taking interface.

I created this visual tutorial in Pages [39], which is primarily a word processor for the macOS, that however provides a handy set of features for working with images. One of these was a sheet-paper imitation border, which was useful for creating the illusion of a paper background.

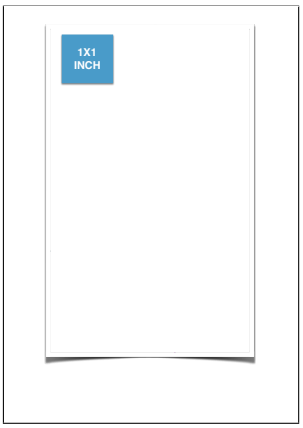
The sorghum representation was created by extracting an image of sorghum going through the image analysis process right after *findContours()*. The contours were then drawn onto a blank canvas using a random function for the colour of the sorghum. A similar process was used to develop the app icon.

5.3 Designing user interface for uploading,processing images

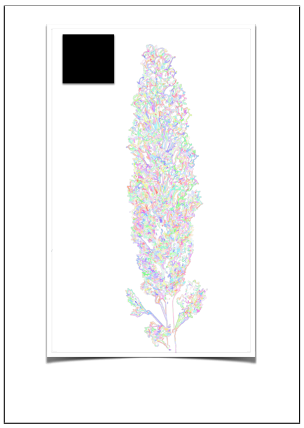
As mentioned, the user has two options of providing images for yield estimation computation. Users can either upload a photography from the device's gallery or capture a photo using the built-in camera. The implementation of this is relatively simple, because iOS provides a common way to access both features. The *UIImagePickerController* class manages a system supplied interface for taking pictures and for choosing saved images. It also manages



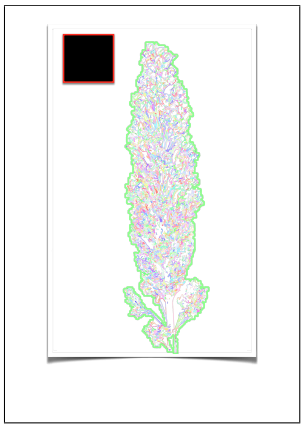
(i) Prepare a sheet of paper



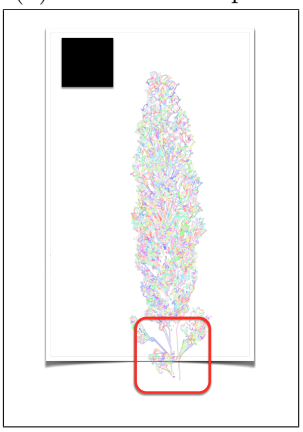
(ii) Draw a solid square



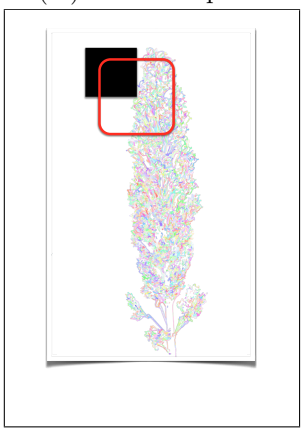
(iii) Place the plant



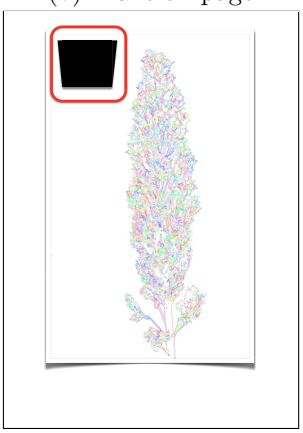
(iv) Valid result



(v) Plant off page



(vi) Plant touches square

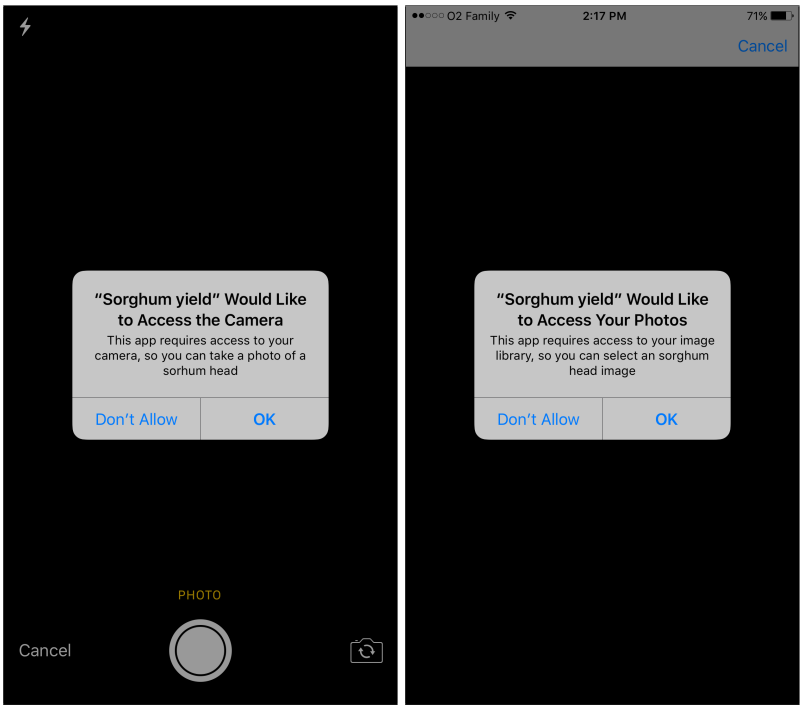


(vii) Sharp angle

Figure 5.3: Tutorial screens in order as they appear in the tutorial

interactions and delivers the results of these interactions to the delegate object[40].

To gain access to the device image gallery or the camera, much like in location services, the device prompts the user to allow these capabilities. Each of these prompts displays an alert with a developer specified text asking the user to either allow or deny app access to the gallery/camera.



(i) Camera consent screen (ii) Gallery consent screen

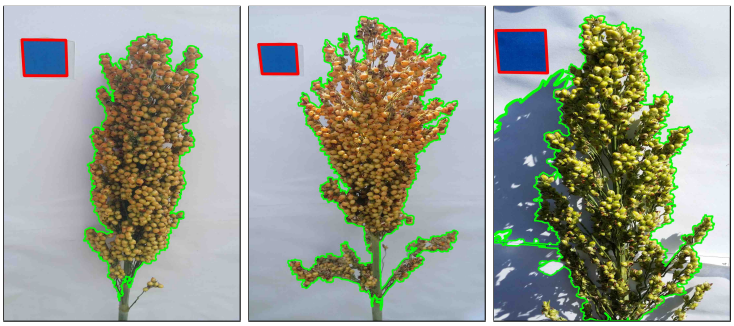
Figure 5.4: Requesting user permission for images

5.3.1 User interface to preview results

This part of the process is essential, because this is the part where the user gets to see the result of the sorghum head analysis. It needs to display the sorghum head area, and the analysed image for the user's approval. The approval is a crucial part of the process, because it lets the user discard unsuccessful or imprecise analysis attempts.

Although great care has gone into designing the algorithm, there will still be cases where the algorithm might misidentify the sorghum head or calculate a wrong head area. Fortunately, the results of the analysis can be displayed in a way that is easily understandable for the user.

Upon viewing these image analysis results the user is presented with two buttons:



(i) Image 1 (ii) Image 2 (iii) Image 3

Figure 5.5: These figures show the result of image analysis performed on sorghum heads

- **"Matches:"** If the red outline matches the square and the green outline matches the plant, such as in images 5.5i and 5.5ii, the user can simply select that the outlines match and the head is added to the list of accepted readings in the previous screen.
- **"No match:"** If either of the outlines does not match the sorghum head or the blue reference square, such as in 5.5iii, the user can select that the head does not match and the reading is discarded.

5.4 Interface for displaying yield results

This is a straightforward process, as all that is needed is to pull all data from our managed data structure in core data, perform calculations on it and then display all sub calculations. Since there is quite a few sub calculations that need to be shown, they might not fit comfortably on smaller devices like the iPhone 4s. Therefore, a scalable user interface element was chosen here, namely *UITableView*[41]. *UITableView* displays a list of items in a single column and allows users to scroll through the table.

We still must consider that the final yield estimation displayed here depends on the seeds per pound component. It was therefore decided that the user will be able to select their seeds per pound variable using a user interface element, that predefines range. Upon every change, the number that displays the final yield estimation in the last column will be updated. Therefore, the user will be able to see how his final yield estimation depends on the final values of seeds per pound.

For this I decided to use a slider, with the values between 9000 and 22500, which is a typical range of seeds per pound for sorghum.

Once the user clicks the "Send report and continue button" the final yield estimation valued based on the value of seeds per pound on slider is stored in the data structure.

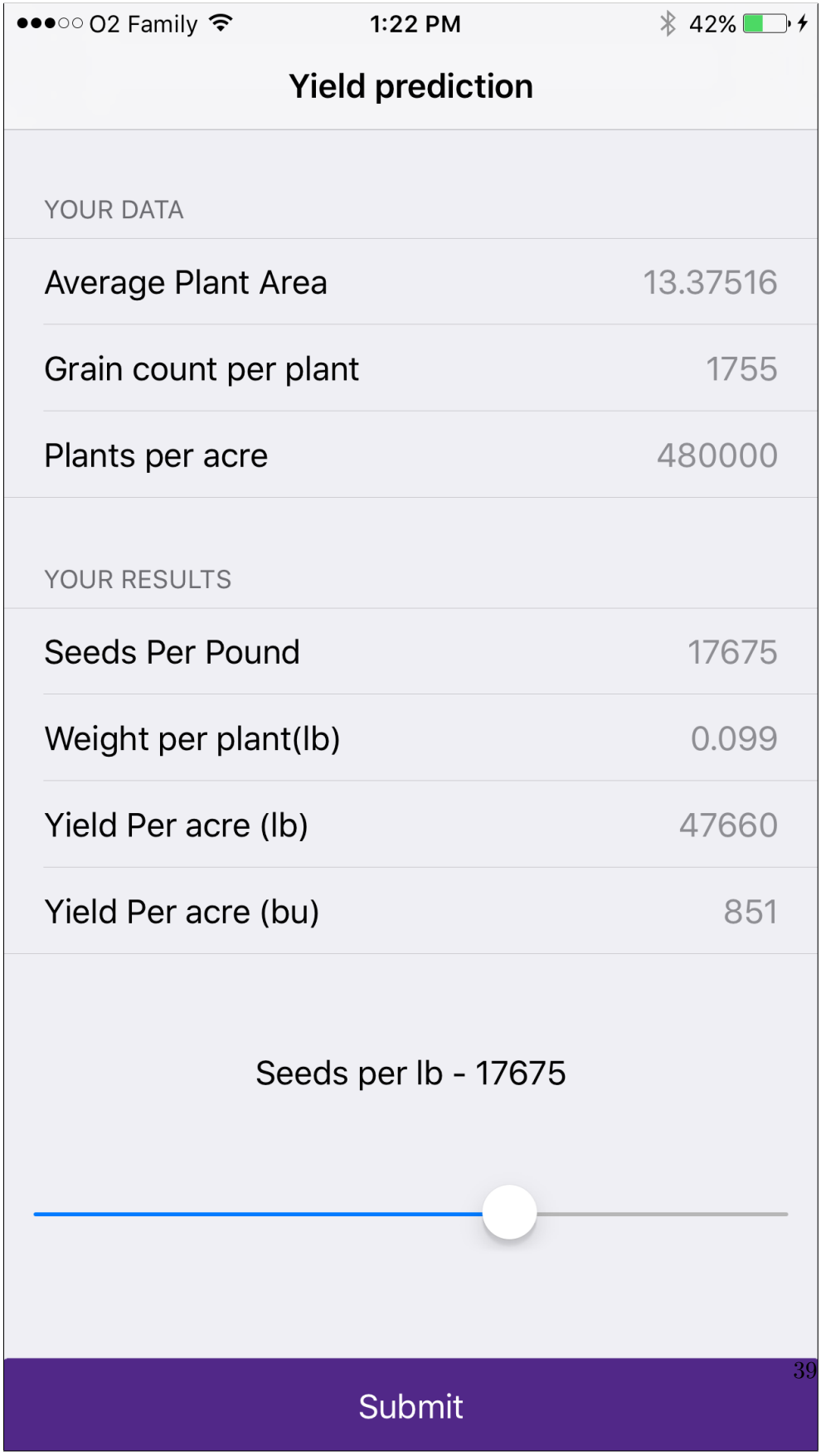


Figure 5.6: Yield result display interface

Clicking the send report button also initiates an upload of the data structure to the backend database. However, the user is prompted first whether he agrees that the results of his yield forecast will be stored. If the user disagrees no data is stored.

5.4.1 Final screen

Since this screen simply statically shows information I chose to implement it using a static table view with static cells. The individual links open in the devices web browser.

5.5 Back end implementation

Firebase stores its data as key-value pairs, where values can have the following JSON types:

- NSString
- NSNumber
- NSDictionary
- NSArray

Since all entries in our *Fieldmeasurement* data structure can be converted to these types, with the exception of the stored images, we can simply store the entire data structure as a dictionary in the database. The key for each dictionary will be created by using the *childByAutoId()* method, provided by firebase, that generates a new child location using a unique key, that is also timestamped to ensure chronological sorting [42]. This ensures that every entry into the firebase will have a unique identifier. [43]

As mentioned images, must be stored differently because firebase database does not support image storage in the real-time database. Fortunately, it provides a cloud storage that serves to store user-generated content, such as photos. For each *FieldMeasurement* a new folder is created in the firebase storage, with the unique ID generated by *childByAutoId()*. Each image is then renamed to the *measurementID* field from "Measurement", which uniquely identifies each image provided by the user and its analysis. That way the researchers can then cross-reference the stored images to the database entry, which specifies how large the analysed sorghum head area was. The images provided by the user are then uploaded into the firebase storage.

As mentioned firebase already provides a web interface for accessing data. Figure 5.8i shows how the interface for viewing database data. Figure 5.8ii shows the folder structure of the cloud storage, where each folder is available for download. As we can see the folder Id's correspond to the Id's from 5.8i.

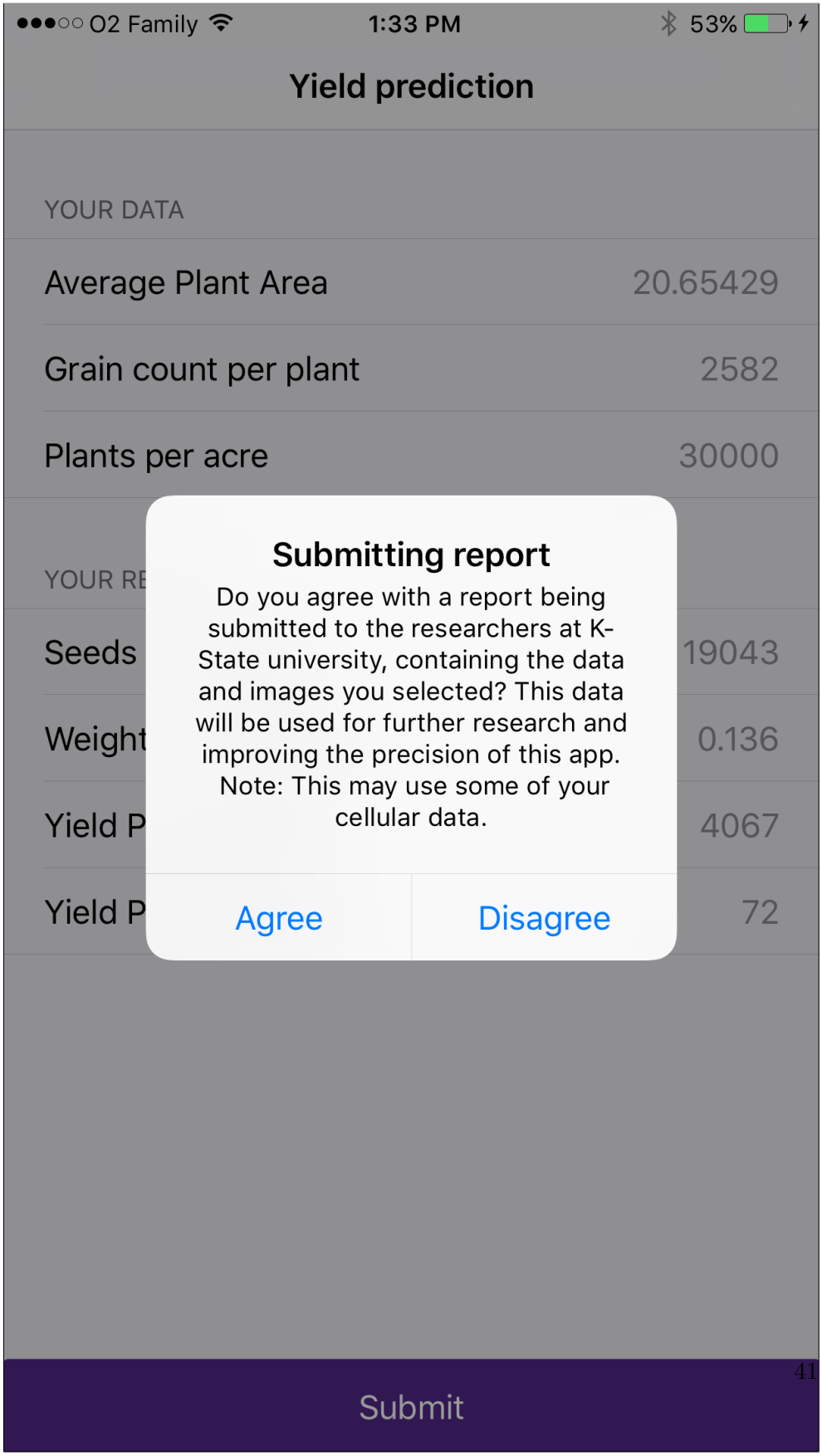
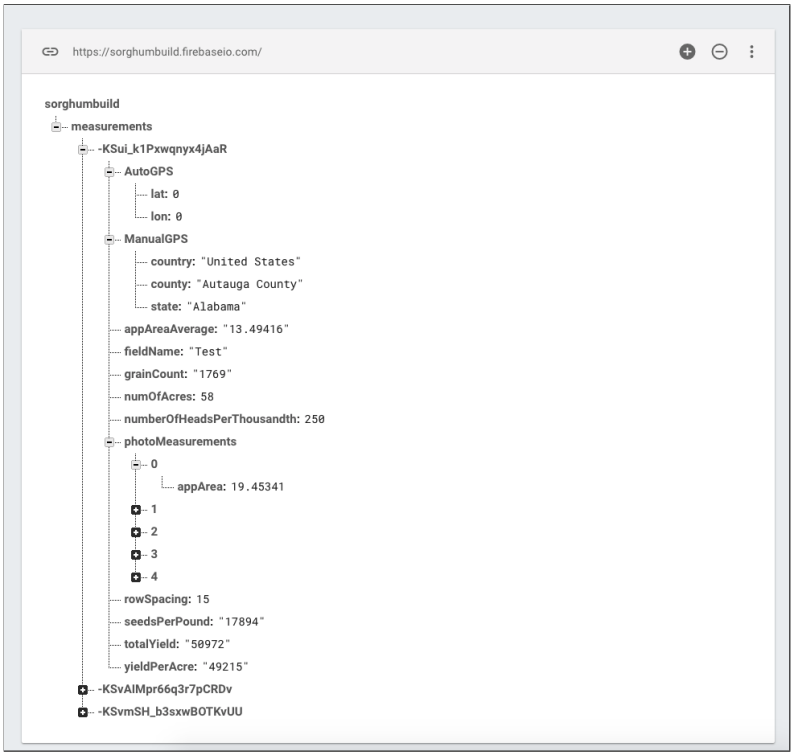
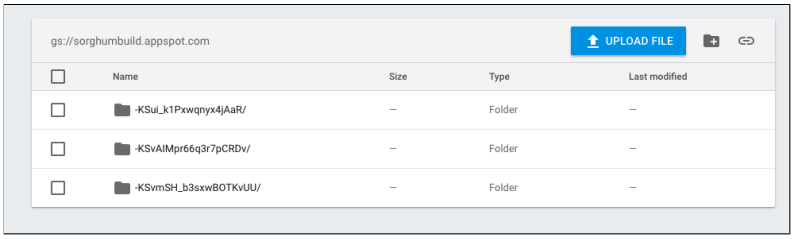


Figure 5.7: Send report window



(i) Database



(ii) Storage

Figure 5.8: Backend interface

Chapter 6

Testing and documentation

6.1 Testing image analysis

Given the large testing data set available, which contained several hundred images, an automatic testing method had to be implemented. One that can process these hundreds of images and organize the results of the image analysis on these images in a systematic way. The result of each algorithm test can then be used to track algorithm accuracy, while under development.

6.1.1 Processing sample data

Since the algorithm is written in C++, which does not have extensive native support for directory browsing, the original data set had to be modified into a more accessible hierarchy.

The original directory tree was set up as follows:

```
BM Ash ..... Researcher name and city abbreviation
├── 101 ..... Field identifier
│   ├── 20150901_141142.jpg ..... Timestamp identifier
│   ├── ...
│   └── 20150901_141517.jpg ..... Timestamp identifier
├── ...
└── 301 ..... Field identifier
    ├── 20150903_163149.jpg ..... Timestamp identifier
    ├── ...
    └── 20150903_163316.jpg ..... Timestamp identifier
```

Along with these images an image data set was provided in the form of a spreadsheet, which had the following format for each top level directory

Since these spreadsheets were provided by researchers who used sophisticated algorithms I took these results as a benchmark to test my results against.

BM ASH					
Images	Plot	Square %	Head %	Head surface $inch^2$	Grain number
1	101	0.0136	0.2974	21.92	2726.78
2	101	0.0111	0.3088	27.77	3391.01
3	101	0.0257	0.2873	11.16	1504.13
. . .	. . .	. . .	. . .	. . .	. . .
8	311	0.0177	0.1727	9.7799	1347.08
9	311	0.0166	0.3290	19.85	2491.36
10	311	0.0169	0.3526	20.91	2611.29

Table 6.1: Original data table

Therefore, I will refer to this data as "true" from now on i.e. "true head surface".

Images 1 through 10 correspond with the time stamped images in the folders. That means that the image taken first is image 1 and so on. To conform the directory structure to the spreadsheet naming convention the entire image set directory had to be renamed to correspond to the spreadsheet identifiers. For that purpose, I wrote the following shell script and executed it on the parent folder of the dataset.

```
#!/bin/sh
for subdir in *; do
    num=1
    for entry in "$subdir"/*
    do
    echo "$subdir/$num"
    mv "$entry" "$subdir/$num.jpg"
    num=$(( num+1 ))
    done
done
mv $subdir/file.txt $subdir.txt; done;
```

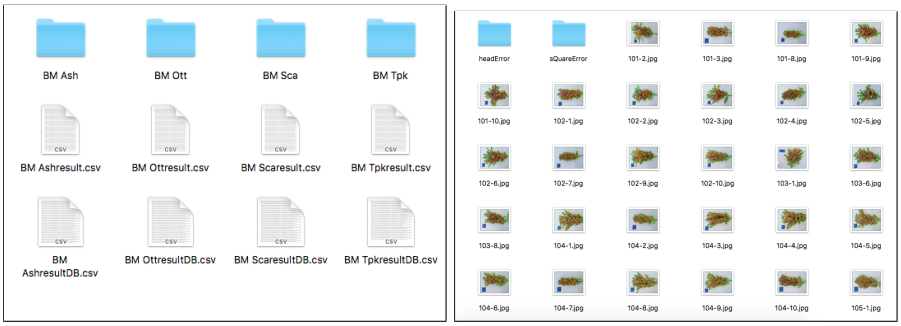
6.1.2 Algorithm testing

To test the image analysis algorithm I wrote a testing program that:

- Iterates through all top directories through sub-directories all the way to the individual images.
- Applies image analysis algorithm and calculates results for:
  - Square percentage
  - Sorghum head surface

- Reads relevant data from data spreadsheet and gets all true results such as true square percentage
- Compares the read values with the calculated ones, writes output for image in spreadsheet
- Saves analysed image into a result directory with same hierarchy as the original image set, if calculated values deviated too much from original values it saves image to a special sub-directory for programmer to inspect
- Measures processing time for each folder and outputs on command line
- Each top directory has a database spreadsheet for previous runs of the algorithm and appends each new run to track algorithm improvement

6.1.3 Processing output data



(i) All top level directories (ii) Contents of BM Ash

Figure 6.1: Result directory hierarchy

So, on each run the results for each image were saved in a spreadsheet for each top level directory. That way a directory like BM Ash had a spreadsheet named "BM AshresultDB" Here is a preview of the spreadsheet after a single run. The most important column is the last one, because it lets us see how the algorithm head size compared to the true head surface. As we can see image 1 in folder 101 was analysed nearly perfectly with only a 6% deviation for square surface and a nigh 0% deviation for head surface. Nevertheless, we are more interested in ways to improve the algorithm and therefore we seek the lines that show clear algorithm failure such as the last line. The spreadsheet representation lets us see at a glance in which images the square was not found. Finding that square in the result image directory is quite effortless as well, because images where square finding failed or deviated too much are stored separately.

Folder Number	Image Number	True SQ %	MY SQ %	SQ Comp	True Head SZ	MY Head SZ	Head comp
101	1	1.86	1.97	1.06	17.31	17.30	1.00
101	2	1.68	1.79	1.07	17.15	17.34	1.01
101	3	1.46	1.55	1.06	21.75	22.31	1.03
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
311	8	1.59	1.44	0.91	26.83	32.76	1.22
311	9	2.28	2.40	1.05	11.27	11.54	1.02
311	10	1.97	0.00	0.00	19.31	inf	inf

Table 6.2: Algorithm result table

As mentioned to track the algorithm improvement progress, every iteration of the algorithm was tested and the results archived. To track the improvement each algorithm iteration was saved as a line in the *DB.csv file for each top level directory.

Resolution scale	Canny Thresh	Dilation Size	Square Errors	Head Errors	Square dif	Head dif
0.5	10	3	107	1	1.02384	1.04
0.5	30	2	15	29	1.01274	0.956461

Table 6.3: Algorithm progress tracking table

Since I realised that the threshold for the canny algorithm and the dilation size of resulting binary image play a huge role in the process of finding the right contours in the image these two variables were stored in the result database for each algorithm run. The changes that were done to the algorithm for that particular run were appended to the database subsequently by hand in order to maintain a better overview of the progress.

This system provided a great starting point for improving the algorithm. We can see that by increasing the canny threshold to 30 the number of squares that were incorrectly identified dropped from 107 to 15 even though the number of errors with heads rose by 28. We can also see that the total square difference across all non-erroneous images in this top-level directory dropped by roughly 1%.

It is interesting to note that the changing dilation size has on calculating surface. A higher dilation size tends to increase the surface calculated and therefore for dilation size 3 we get an overestimation of head surface compared to true values.

Of equal importance are the ”Square errors” and ”Head Errors” columns. The square reading is identified as faulty, when the difference between the true square percentage and my square percentage deviate by more than 10%. For head errors this threshold is 30%. The results of faulty readings do not count

towards overall head difference, because faulty values would influence the final value too much. In general, we can rely on the fact that head errors and square errors will be visible to the user, during algorithm output inspection in the application and will be discarded.

6.1.4 Testing results

Over the course of testing I ran the algorithm hundreds of times for different values and algorithm designs. The individual folders that the dataset was split into represented the fields that the samples were from. Each field/ folder had noticeably different results when testing the algorithm. The best folder by far was the "BM ASH" folder which represents an area known as Ash Valley in Kansas. The folder contains 330 images. On this dataset I was able to get these results:

- Square Errors: 12
- Head Errors: 1
- Square dif :1.05035
- Head dif: 1.00339

That means that the algorithm only failed on 13 out of 330 images.

The worst dataset is "BM TPK", which represents the area of Topeka Kansas. On this dataset I got about 14 square errors and 32 head errors, because the images were taken under direct sunlight, which caused a lot of shadows to be picked up as parts of sorghum heads and therefore increased the area over the 30% threshold. The deviation of true head size compared to my head size was also above 13% for the dataset.

6.2 Testing iOS app

Unfortunately, the amount of automated testing that can be done on the app is limited. We could perform automatic unit testing, but since the scope of the app is quite limited, automated testing does not make much sense.

It is however, very important that the image analysis be tested on the iOS app. Image analysis testing was done on images taken by professional researchers and were therefore of very high quality. The app needs further testing in real-life conditions and among real users, to see how users interact with it. It remains to be seen whether users find the process intuitive and whether the image analysis performs well on images taken under different conditions than the images that were supplied for testing.

As the leader of this project, Dr. Ignacio Ciampitti and his team have taken it upon themselves to test the app on further sorghum heads in varied

conditions and show the app to sorghum farmers to see how they respond to it. To make it possible for Dr. Ciampitti and his team to test the application, I had to find a suitable way to distribute the application to them and their test users, that does not require any technological know-how to distribute the application.

There are currently several options for distributing iOS applications for testing. Besides the native TestFlight [44], there are also 3rd party solutions like Hockey App [45] by Microsoft or Fabric [46] by Twitter.

To give the researchers access to the latest version of the app without submitting it to the App Store, I decided to go with TestFlight, which is an online service for over-the-air installation of mobile applications. To understand the reasons why TestFlight was the best option, I must delve into the process behind submitting iOS applications.

6.2.1 Submitting sorghum yield

There are four levels of developer accounts, with a varying degree of access and capabilities. To even develop for the iOS platform, an Apple ID is needed. With this you can develop your app and test on devices by building from XCode. There is however no way to distribute the application without having one of the three paid tiers of developer accounts. The three tiers are[47]:

- Individual
- Organization
- Enterprise Program

The only developer account that could be acquired by the researchers was the Individual one, because organization and Enterprise Program require a DUNS number, which is a unique nine-digit identifier only given to businesses[48].

There are only two options to choose from when it comes to distributing iOS applications using an Individual developer account. You can either select the Ad Hoc option or go through the App store. Since we only want to test the application and therefore do not want to submit it to the App Store yet.[49]

The major issue with Ad Hoc distribution is that all devices that the app will be installed on, need to have their UDID number added to the provisioning profile devices list. The UDID number is a unique device identifier for each iOS device. Every time an App is installed on a device the device first checks, whether its UDID is included in the device list for the Ad Hoc provisioning profile.[49]

All this simply means, every time the researchers would want to install the application on a new test device, they would have to add the device to the devices list in the provisioning profile. This is also true for distribution using

either HockeyApp or Fabric. The process of retrieving the UDID and adding it to the provisioning profile devices list is quite complicated.

Fortunately, TestFlight solves this issue, by using an App Store Provisioning profile, without actually submitting to the App Store. It takes care of registering devices automatically and all you need to do, is send an email invite to your testers. The only downside is that they must install the TestFlight app first, but that is relatively easy process, since TestFlight is simply installed through the App store. [49]

Therefore, I decided to go with distributing the testing builds using TestFlight. The latest build was uploaded to the TestFlight platform using the Dr. Ciampitti's developer account, which gives him access to invites and feedback.

6.3 Documentation

As mentioned previously, this is a student project that is likely to be passed on to other people after my graduation. Therefore, it is necessary, to make the transition for other programmers as effortless as possible. To achieve that proper code documentation is important.

Since Xcode 8 introduced automatic comment stub generation [50], i decided to use this feature. Using the shortcut *command+option+/* Xcode 8 now generates a comment stub.

```
/**
 Loads lines from files

 @param fileName  Name of the file
 @param fileType  Type of the file
 @param directory Directory of the file
 @param encoding  Encoding used
 @param delimeter Delimeter used to separate data

 @return Returns an array of string lines
 */
- (NSArray * ) getLinesFromFile:  (NSString *)fileName ofType :(
    NSString * ) fileType inDirectory :(NSString *) directory
    encodedWith :(NSStringEncoding) encoding usingDelimeter:(
    NSString *) delimeter {

    NSString* filePath = [[NSBundle mainBundle] pathForResource:
        fileName ofType:fileType];

    NSString *fileContents = [NSString stringWithContentsOfFile:
        filePath encoding: encoding error:NULL];
    return [fileContents componentsSeparatedByString:delimeter];

}
```

./tex/comment

As we can see in lines 4-8 you can describe individual parameters. Line 10 shows that you can also comment the return values of functions. Line 2 shows a general description of what the function is used to achieve.

In figure 6.2 we can see that XCode actually uses these auto-generated comments to generate a function description that a programmer can use when using the function somewhere in the code. This lets the programmer see at a glance what the function is used for, what to pass as parameters and what to expect to be returned.

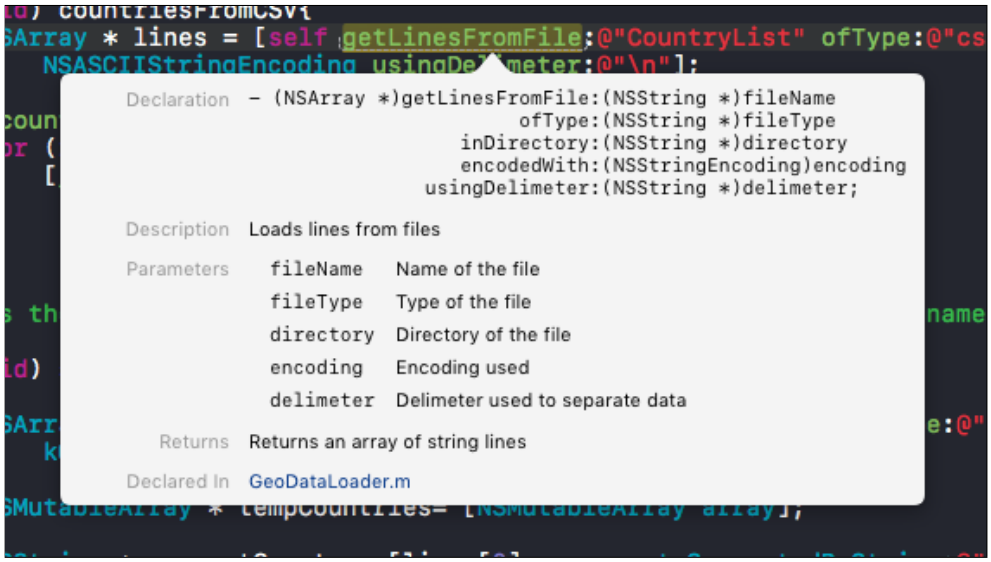


Figure 6.2: XCode function description

I used appledoc to auto-generate a html documentation from these comments. Appledoc is command line tool used to generate Apple-like source code documentation from specially formatted source code comments[51].

Conclusion

The goal of this thesis was to analyze, design and implement an iOS application and an image analysis algorithm, which analyses the area of the plant sorghum. By combining these two parts, we can make a product that lets a user estimate the yield forecast of their sorghum fields.

The biggest accomplishment of this thesis, is in my opinion the implementation of the image analysis algorithm capable of analysing images, which contain a blue square of specified size and a sorghum head. The algorithm peforms a head area estimation, which is then used in a yield forecasting algorithm to predict the yield of a sorghum field. This algorithm was tested on roughly 1500 images and failed to find the blue square on only 86 valid images. The averaged head surface deviation of non faulty readings compared to dataset values was roughly 5%.

The presented algorithm and application do not implement many improvements that I thought of, while working on this thesis. One feature that I think would improve the application noticeably is real time image analysis using the camera in video mode. That means the user would simply hover the camera over the plant, and the analysis would be performed in real time, scanning multiple times per second. While this is technically possible, the algorithm performance would have to be improved.

Another big improvement would be rewriting the iOS application in the Swift programming language, to prepare the application for future modifications. Swift is swiftly taking over objective-C in terms of market share, and is presumably going to be the dominant language used to write iOS applications in the future.

I plan to continue working on the project and taking into consideration the feedback I get from the researchers at Kansas State University. Eventually I would like to implement the improvements I mentioned and others, perhaps as part of my Master thesis.

Bibliography

[1] Yi-Hong Wang, Hari D. Upadhyaya and Chittaranjan Kole. *Genetics, Genomics and Breeding of Sorghum (Genetics, Genomics and Breeding of Crop Plants)*. CRC Press, 2014. ISBN: 1482210088.

[2] *Sorghum production*. `http://faostat3.fao.org`. Accessed: 2016-8-28.

[3] Jeff Dahlberg John H. Wiersema. 'The Nomenclature of Sorghum bicolor (L.) Moench (Gramineae)'. In: *Taxon* 56.3 (2007), pp. 941–946. ISSN: 00400262. URL: `http://www.jstor.org/stable/25065876`.

[4] Food and Agriculture Organization of the United Nations. *Sorghum and Millets in Human Nutrition (FAO Food and Nutrition Series)*. FAO, 1995. ISBN: 9251033811.

[5] Board on Science et al. *Lost Crops of Africa: Volume I: Grains (Lost Crops of Africa Vol. I)*. National Academies Press, 1996. ISBN: 0309049903.

[6] Wigmore Ivy. *precision agriculture*. URL: `http://whatis.techtarget.com/definition/precision-agriculture-precision-farming` (visited on 12/09/2016).

[7] G.R. Spinks. 'Uses and Methods of Crop Forecasting'. In: *Review of Marketing and Agricultural Economics* 24.01 (1956). URL: `https://ideas.repec.org/a/ags/remaae/8933.html`.

[8] Robin B Matthews and William Stephens. *Crop-Soil Simulation Models: Applications in Developing Countries*. CABI, 2002. ISBN: 0851995632.

[9] K-State Extension Agronomy and Steve Watson. *eUpdate, Issue 473*. URL: `webapp.agron.ksu.edu/agr_social/eupdates/eUpdate090514.pdf` (visited on 29/12/2016).

[10] K-State Extension Agronomy and Steve Watson. *eUpdate, Issue 474*. URL: `https://webapp.agron.ksu.edu/agr_social/eu_article.throck?article_id=344` (visited on 29/12/2016).

[11] *Computer Vision platform using Python.* URL: http://simplecv.org/ (visited on 29/12/2016).

[12] liuliu. *ccv.* URL: http://libccv.org/ (visited on 29/12/2016).

[13] URL: https://commons.wikimedia.org/wiki/File:OpenCV_Logo_with_text.png (visited on 29/12/2016).

[14] Adrian Kaehler and Gary Bradski. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library.* O'Reilly Media, 2016. ISBN: 1491937998.

[15] *Using C++ With Objective-C.* URL: http://web.archive.org/web/20101203170217/http://developer.apple.com/library/mac/#/web/20101204020949/http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ObjectiveC/Articles/ocCPlusPlus.html (visited on 29/12/2016).

[16] Turkmen Suleyman. *squares.cpp.* URL: https://github.com/opencv/opencv/blob/master/samples/cpp/squares.cpp (visited on 29/12/2016).

[17] *Image Filtering.* URL: http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html (visited on 29/12/2016).

[18] *Operations on Arrays.* URL: http://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html (visited on 29/12/2016).

[19] *Canny Edge Detector.* URL: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours (visited on 29/12/2016).

[20] Ong Josh. *Lowe's deploying 42,000 iPhone-based POS systems in retail overhaul.* URL: http://appleinsider.com/articles/11/09/08/lowes_deploying_42k_iphone_based_pos_systems_in_retail_overhaul (visited on 29/12/2016).

[21] Cowart Jim. *Pros and Cons of the Top 5 Cross-Platform Tools.* URL: https://www.developereconomics.com/pros-cons-top-5-cross-platform-tools (visited on 29/12/2016).

[22] Marszałek Krzysztof. *NATIVE VS HYBRID - DEMYSTIFYING THE TECHNOLOGY DILEMMA.* URL: http://howwedostartups.com/articles/Native-vs-Hybrid (visited on 29/12/2016).

[23] *UserDefaults.* URL: https://developer.apple.com/reference/foundation/userdefaults (visited on 29/12/2016).

[24] Damjanović Ivan. *Why Realm is great and why are we not using it.* URL: http://bsktapp.com/blog/why-is-realm-great-and-why-are-we-not-using-it/ (visited on 29/12/2016).

[25] *What Is Core Data?* URL: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/ (visited on 29/12/2016).

[26] *realm*. URL: https://github.com/realm/realm-cocoa (visited on 29/12/2016).

[27] URL: https://firebase.googleblog.com/2016/05/firebase-expands-to-become-unified-app-platform.html (visited on 29/12/2016).

[28] *App success made simple*. URL: https://firebase.google.com/ (visited on 12/09/2016).

[29] *OpenCV iOS - Image Processing*. URL: http://docs.opencv.org/2.4/doc/tutorials/ios/image_manipulation/image_manipulation.html (visited on 29/12/2016).

[30] *Canny Edge Detector*. URL: http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html (visited on 29/12/2016).

[31] *iPhone 7*. URL: http://www.apple.com/iphone-7/specs/ (visited on 29/12/2016).

[32] *Basic Structures*. URL: http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html (visited on 29/12/2016).

[33] *Miscellaneous Image Transformations*. URL: http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html (visited on 29/12/2016).

[34] *UIKeyboardTypeDecimalPad*. URL: https://developer.apple.com/reference/uikit/uikeyboardtype/uikeyboardtypenumberpad?language=objc (visited on 29/12/2016).

[35] Leybaert Philippe. *Types in objective-c on iPhone*. URL: http://stackoverflow.com/questions/2107544/types-in-objective-c-on-iphone (visited on 29/12/2016).

[36] *Reduce Location Accuracy and Duration*. URL: https://developer.apple.com/library/content/documentation/Performance/Conceptual/EnergyGuide-iOS/LocationBestPractices.html (visited on 29/12/2016).

[37] *Infographic: Cracking the Code on Location Services On*. URL: http://blog.skyhookwireless.com/infographic-cracking-the-code-on-location-services-on (visited on 29/12/2016).

[38] *Page View Controllers*. URL: https://developer.apple.com/library/content/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/PageViewControllers.html (visited on 29/12/2016).

[39] *Documents that stand apart. Created together*. URL: http://www.apple.com/pages (visited on 29/12/2016).

[40] *UIImagePickerController*. URL: https://developer.apple.com/reference/uikit/uiimagepickercontroller (visited on 29/12/2016).

[41] *UITableView*. URL: https://developer.apple.com/reference/uikit/uitableview (visited on 29/12/2016).

[42] *FirebaseDatabase Framework Reference*. URL: https://firebase.google.com/docs/reference/ios/firebasedatabase/api/reference/Classes/FIRDatabaseReference (visited on 29/12/2016).

[43] *Firebase Realtime Database*. URL: https://firebase.google.com/docs/database/ (visited on 29/12/2016).

[44] *TestFlight Beta Testing*. URL: https://developer.apple.com/testflight/ (visited on 29/12/2016).

[45] *HockeyApp - The Platform for Your Apps*. URL: https://hockeyapp.net (visited on 29/12/2016).

[46] *Built by the same team that built Crashlytics*. URL: https://get.fabric.io/ (visited on 29/12/2016).

[47] *Choosing a Membership*. URL: https://developer.apple.com/support/compare-memberships/ (visited on 29/12/2016).

[48] *What's a D&B*. URL: http://www.dnb.com/duns-number.html (visited on 29/12/2016).

[49] *Exporting Your App for Testing*. URL: https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/TestingYouriOSApp/TestingYouriOSApp.html (visited on 29/12/2016).

[50] *What's New in Xcode*. URL: https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/WhatsNewXcode/introduction.html (visited on 29/12/2016).

[51] *About appledoc*. URL: https://github.com/tomaz/appledoc (visited on 29/12/2016).

Appendix A

Glossary

FAOSTAT Food and Agriculture Organization Corporate Statistical Database

OpenCV Open Source Computer Vision Library

HSV Hue-Saturation-Value model

GPS Global Positioning System

CPT Cross platform tools

NoSQL Not only SQL

SQL Structured Query Language

JSON JavaScript Object Notation

Appendix B

Contents of enclosed CD

- `readme.txt` ..... File with CD contents description
- `SorghumYield` ..... Directory of the iOS project
  - `SorghumYield.xcworkspace` ..... XCode project workspace
- `SorghumTest` ..... Testing environment
  - `OpenCVTest` ..... Source code directory
  - `SampleData` ..... Directory with sample data
    - `BM Ott` ..... Image data directory
    - `BM Ott.csv` ..... Result data csv file
- `Documentation` ..... Directory containing generated code documentation
- `thesis` ..... the directory of LaTeX source codes of the thesis