5-2013

On Mobile Detection and Localization of Skewed Nutrition Facts Tables

Christopher Blay
Utah State University

ON MOBILE DETECTION AND LOCALIZATION OF SKEWED NUTRITION FACTS TABLES

by

Christopher Blay

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Dr. Vladimir A. Kulyukin Major Professor	Dr. Curtis Dyreson Committee Member
Dr. Nick Flann Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2013

ABSTRACT

On Mobile Detection and Localization of Skewed Nutrition Facts Tables

by

Christopher Blay, Master of Science

Utah State University, 2013

Major Professor: Dr. Vladimir A. Kulyukin
Department: Computer Science

With about 3.6 million adults in the United States having visual impairment or blindness, assistive technology is essential to give these people grocery shopping independance. This thesis presents a new method to detect and localize nutrition facts tables (NFTs) on mobile devices more quickly and from less-ideal inputs than before. The method is a drop-in replacement for an existing NFT analysis pipeline and utilizes multiple image analysis methods which exploit various properties of standard NFTs.

In testing, this method performs very well with no false-positives and 42% total recall. These results are ideal for real-world application where inputs are analyzed as quickly as possible. Additionally, this new method exposes many possibilities for future improvement.

(54 pages)

PUBLIC ABSTRACT

On Mobile Detection and Localization of Skewed Nutrition Facts Tables

Christopher Blay

The Computer Science Assistive Technology Laboratory (CSATL) at Utah State University has a long history of research in visually impaired grocery shopping tech. CSATL's ShopMobile II introduced nutrition facts table (NFT) analysis but only with perfectly aligned and square input images.

A new method which detects and localizes NFTs more quickly and from rotated or non-square images has been released and is slated for integration with ShopMobile II to improve this feature substantially. This is great news for the estimated 3.6 million adults in the United States having visual impairment or blindness and also opens the doors to other applications where analysing NFTs can significantly aid users.

By combining image analysis methods in creative ways, the new method avoids detecting NFTs where there are none and properly locates them in about 42% of images. This is remarkable considering images will be processed as quickly as possible on the device – a standard Android smartphone. The CSATL isn't stopping here though: the new method exposes several possibilities for further improvements.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Reading a Nutritional Facts Table (NFT) is not a simple task – even for humans. In *Eye tracking and nutrition label use: A review of the literature and recommendations for label enhancement*, a significant amount of research concerning readability of NFTs is reviewed, ultimately resulting in suggestions for improvement [5]. These suggestions include items as simple as increasing the surface area of NFTs and reducing visual clutter around NFTs.

Given the difficulty regular people have reading NFTs, now imagine the hurdle visually impaired (VI) shoppers face while selecting foods based on their nutritional content. In a report published by Prevent Blindness America and the National Eye Institute, at least an estimated 3.6 million adults in the United States have visual impairment or blindness. Among those, over 1.4 million people are legally blind [2].

According to Helal et al. there are five steps which take place in VI grocery shopping: travelling to the store, shopping for the desired product, making a payment, leaving the store, and travelling back home [6]. VI individuals already have a great deal of freedom in travelling to various places but have traditionally been limited to relying upon a sighted individual while shopping for desired products.

For these VI shoppers, only assistive technology can grant the same level of freedom as that of a sighted shopper. This is not a new idea; research already shows that assistive shopping systems help VI shoppers shop independently [13]. GroZi [17], iCare [8], Trinetra [15], RoboCart [9], and ShopTalk [11] are assistive shopping systems which go about solving this complicated problem in different ways. However, these solutions rely on highly custom hardware and/or shopping environments which makes their implementation costly and their maintenance troublesome.

Solutions such as ShopMobile [10] and ShopMobile II [12] attempt to overcome these

barriers by combining barcode scanners with an existing database of product data. Using a barcode scanner alone also only goes so far though; these databases can have old, missing, or incomplete entries. How are new entries added? Who keeps existing entries updated? What kind of infrastructure is required and how is it funded?

An assistive shopping system which could detect, localize, and extract data from NFTs using off-the-shelf hardware would certainly be an improvement in this field. All information would be directly taken from the real NFT; the same one read by sighted shoppers. In this paper, we will present a method to perform the detection and localization of NFTs on a smartphone.

Chapter 2 contains a review of related work. Chapter 3 details the image analysis methods which will be employed. Chapter 4 presents our method itself. Chapter 5 explains a trigonometric function used within our method. Chapter 6 discusses our results and their analysis. Finally, Chapter 7 concludes this paper and briefly covers future work.

CHAPTER 2

RELATED WORK

As mentioned in Chapter 1, assistive shopping systems which have already been built with the purpose of aiding VI grocery shopping include GroZi [17], iCare [8], Trinetra [15], RoboCart [9], ShopTalk [11], ShopMobile [10], and ShopMobile II [12]. All of these with the exception of GroZi, iCare, and Trinetra were developed within the Computer Science Assistive Technology Laboratory (CSATL) at Utah State University (USU).

2.1 GroZi

GroZi [17] was developed by the University of California San Diego and published in 2007. It utilizes a custom hardware solution which users can use at a grocery store to "scan" products based on their recognizable front face images. A predetermined list of desired products is used to alert the user when a match is found. Matching is performed with a large database of images stored within the hardware. This database is seeded with *in vitro* images taken in perfect conditions and *in situ* images taken during similar shopping expeditions.

Although this system is primarily meant to aid VI shoppers with simply finding a product they desire instead of obtaining nutritional facts data, it could easily be extended to include extra information within its image database. A limitation of this approach is the requirement to keep the image database updated properly – even with multiple images for the same product.

2.2 iCare

iCare [8] was developed by Arizona State University and published in 2008. Its hardware consists of a PDA using Bluetooth for local communication and WiFi to make queries

and a RFID reader embedded within a glove. It provides a great amount of VI shopping aid at the cost placing RFID tags all over the store and even on each item for sale. A user could simply move the RFID reader near a product or aisle and information would immediately be queried from a local store database to retrieve relevant details.

Similar to GroZi, this system wasn't specifically built with nutritional data retrieval in mind but could be extended to do so quite easily. Unfortunately, the sheer cost and maintenance required to place RFID tags throughout the store and upon all merchandise makes this solution very unappealing to stores. Additionally, detailed product placement information is generally closely guarded which makes the possibility of an accessible store-maintained database unlikely.

2.3 Trinetra

Trinetra [15] was developed by Carnegie Mellon University and published in 2007. Its hardware consists of a Nokia smartphone, Bluetooth wireless headset, and Bluetooth RFID/Barcode scanners. In addition to using RFID similar to iCare, Trinetra can use barcodes on products as well which simplifies the setup and maintenance for the store. It features a caching hierarchy so that devices use local information first and only query centralized databases when needed.

This is another system which could be extended to provide nutritional facts information to VI users but suffers from the same limitations as iCare: some RFID tags must be installed and a store-maintained database is not generally available.

2.4 RoboCart

RoboCart [9] was developed by CSATL at USU and was published in 2005. Its hardware consists of a custom robotic shopping cart which can drive itself through a store to specific locations using specially placed RFID tags at the aisle ends. It also uses a laser range-finder to detect obstacles before running into them and has a wireless barcode scanner which can be used after arriving at a specific location.

The largest issue with this solution is the cost of the robotic cart itself. It contains many expensive parts and is very costly to build or maintain. Additionally, RFID tags must be placed at each aisle and a store-maintained database must be accessible.

2.5 ShopTalk

ShopTalk [11] was developed by CSATL at USU after RoboCart and was published in 2007. The costly robotic cart was abandoned in favor of a wearable computer and keyboard attached to a backpack along with a handheld barcode scanner. It utilized the Modified Plessey (MSI) shelf barcodes rather than individual product barcodes to ascertain location within the store and used a barcode connectivity matrix to guide the VI shopper from one product area to another.

Unfortunately, building the necessary barcode connectivity matrix requires access to a store-maintaned database which is not generally available.

2.6 ShopMobile

ShopMobile [10] was developed by CSATL at USU after ShopTalk and was published in 2010. It uses a standard smartphone with a special case and custom barcode software. The case is designed so that the phone will fit against a shelf edge and read the MSI shelf barcodes. In this fashion, it achieves the same goal that ShopTalk does with the advantage of using more common hardware.

2.7 ShopMobile II

ShopMobile II [12] was developed by CSATL at USU after ShopTalk as was published in 2010. It continues to use a standard smartphone for hardware and now features an eyes-free barcode scanner, a tele-assistance module, and an optical character recognition (OCR) module. It can read both MSI and UPC barcodes with the eyes-free scanner to query nutritional information for a product. The tele-assistance module lets a VI user get sighted help from a family member or friend at a computer. The OCR module also provides NFT information directly from a product.

CHAPTER 3

IMAGE ANALYSIS METHODS

The following image analysis methods are used throughout this localization algorithm. The methods will now be explained in detail so that they can be quickly referred to in Chapter 4 without the need to explain their exact workings again.

One important concept to explain before continuing is that of the image coordinate plane used. Normally the origin of a coordinate plane is in the lower left corner, the x-axis extends positively to the right, and the y-axis extends positively to the top. However, within an image coordinate plane, the origin it located in the upper left corner, the x-axis extends positively to the right, and the y-axis extends positively to the bottom. This means that a smaller x-axis value is closer to the left side of an image and a smaller y-axis value is closer to the top of an image.

3.1 Canny Edge Detector

Reliably detecting edges in an image is a powerful way to detect the features it contains. A good edge detector transforms even the most complex of images into a simple bitmap which still relays important data and simplifies further processing. Within this NFT localization algorithm, the Canny Edge Detector is used to perform edge detection. John F. Canny published *A computational approach to edge detection* in 1986 and while it was a vast improvement over existing edge detectors at the time, it continues to be one of the best edge detection algorithms to this day. This is certainly due to its use of variational calculus which helps distinguish between real edges and noise within an image [4].

The Canny Edge Detector utilizes two thresholds which control the overall amount of edges detected. The thresholds used by our implementation are currently hard-coded and do not change based on the input conditions which can certainly lead to reduced quality

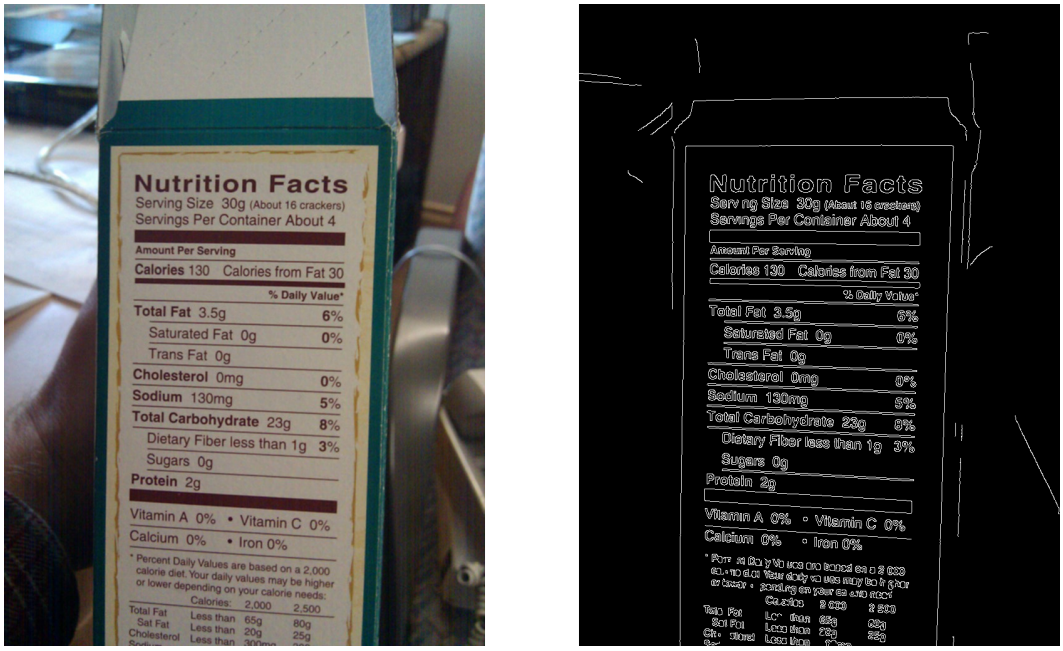

Figure 3.1. Canny Edge Detector Input and Result

of results in poor-quality lighting conditions. It could be useful to monitor Canny edge detection results and slowly adjust the thresholds to optimize the amount of edges located during a given session.

3.2 Hough Transform

Once edge detection data has been obtained, the Hough Transform performs the hard work of extracting information that can be used by the NFT localization algorithm. This powerful technique was published by Richard Duda and Peter Hart in 1972 and is capable of quickly locating paths in a binarized image which follow a generalized polynomial. It works by preparing a pre-determined number of bins for each possible combination of equation inputs and then by incrementing the appropriate bins for each "true" pixel. This novel approach allows the image to be scanned only once and provides a memory usage verses accuracy trade-off [3].

The most basic of polynomials, a line, is generally represented by the equation $f(x) = mx + b$ where the two variables m and b control the slope and y-intercept of the line, respec-

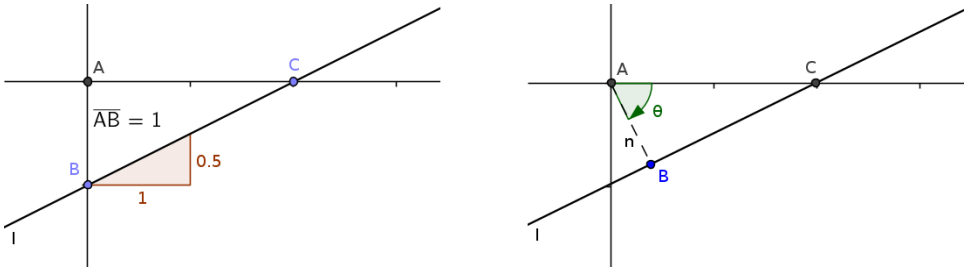

Figure 3.2. The Same Line Defined Within Cartesian- and Polar-Coordinate Systems

tively. However, this form is lacking in one very important way: it cannot easily represent perfectly vertical lines where $m = \infty$. To avoid this condition, the Hough Transform deals with lines defined not within Cartesian coordinate space but rather Polar coordinate space where the analogue linear equation is $r(\phi) = \rho \sec(\phi - \theta)$. Now ρ and θ define the line where ρ is the distance from the line to the origin and θ is its rotation about the origin. A line with $\rho = \theta = 0$ is vertical passing through the origin.

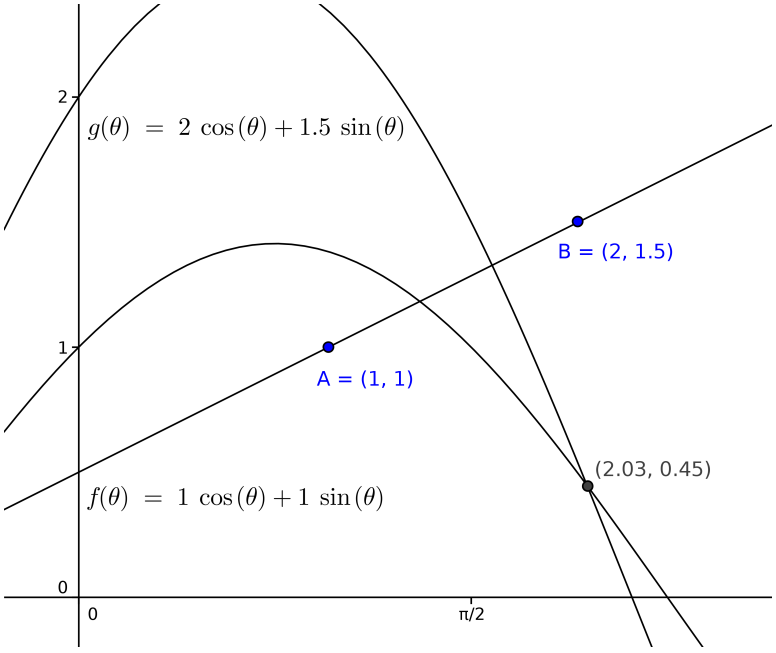

Figure 3.3. Hough Transformation Higher-Level Concept

Figure 3.2 demonstrates how the two systems define the same line. The left graph is in Cartesian coordinate space using $f(x) = mx + b$ where $m = 0.5$ is the slope as demonstrated

by the "rise over run" and $b = -1$ is the y-axis intersection. The right graph is in Polar coordinate space using $r(\phi) = \rho \sec(\phi - \theta)$ where $\rho \approx 0.88$ is the length of the normal and $\theta \approx 63.8$ deg is the clockwise angle from the positive x-axis to the normal.

The higher-level concept of the Hough Transform is that each "true" pixel has a function of all possible ρ, θ combinations defined by $f(\theta) = x \cos\theta + y \sin\theta$ and we are looking for the place where all these functions intersect the most. Figure 3.3 demonstrates this concept by plotting the functions $f(\theta)$ and $g(\theta)$ for points A and B, respectively. The functions both pass through point $C = (2.03, 0.45)$ which means the line passing through A and B is defined in polar coordinate space as $\theta = 2.03$ radians and $\rho = 0.45$.

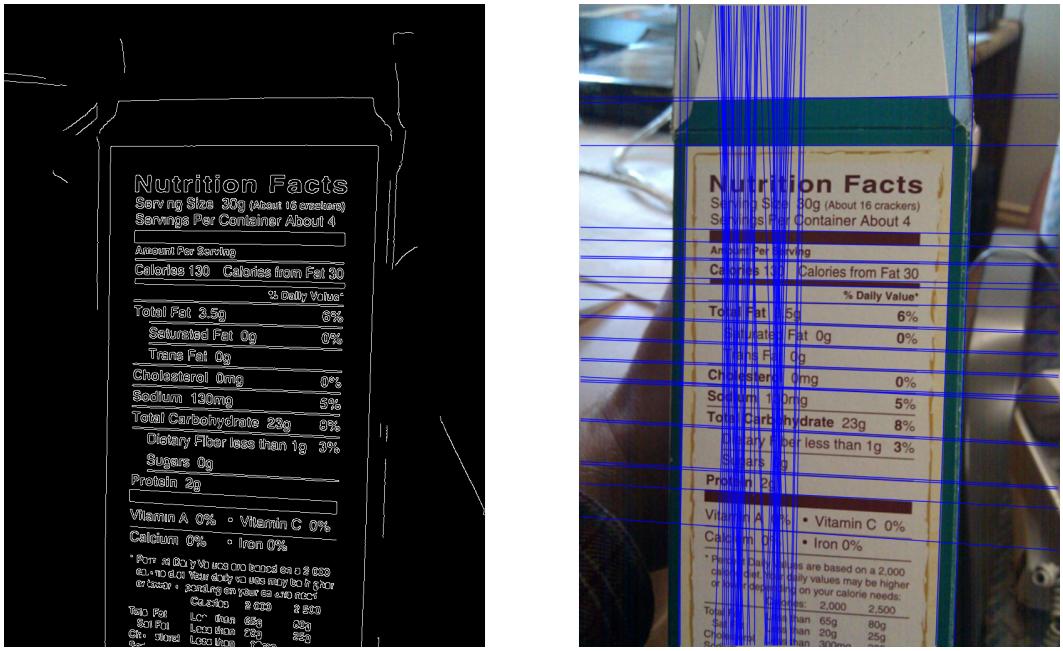

Figure 3.4. Hough Transform Input and Result

To reiterate, ρ and θ are segmented into discrete sets which are combined to create a counter for each unique combination. Then each bin which could match a "true" pixel is incremented. A line is detected if its counter is ultimately greater than some threshold. Figure 3.4 demonstrates how the Hough Transform is used by this method to detect straight lines from Canny Edge Detector results.

While the Hough Transform works very well for detecting lines, the memory required

for more complex polynomials increases exponentially as more variables must be segmented and assembled into every possible combination. This can be alleviated to some extent by lowering the resolution of one or more of the variables in question. Additionally, as we will demonstrate later, the particular way in which Hough Transform is used by this NFT localization algorithm is amenable to reducing memory usage by only allocating a subset of all possible combinations since we are only interested in lines which happen to fit within certain specifications. Further advances can be made in this direction such as per-bin thresholds which would allow for better results in non-square images.

3.3 Dilate / Erode Corner Detection

One of the key components of this NFT localization algorithm involves analysis of corners within the image. Specifically corners formed by text which happens to contain many distinct corners and a significant contrast between the foreground and background colors. A particular corner detection algorithm which uses a combination of various erode and dilate image filters has been determined to produce excellent results for matching corners within text. While it is possible that other corner detection algorithms may produce similar results somewhat faster, this area of research has yet to be explored.

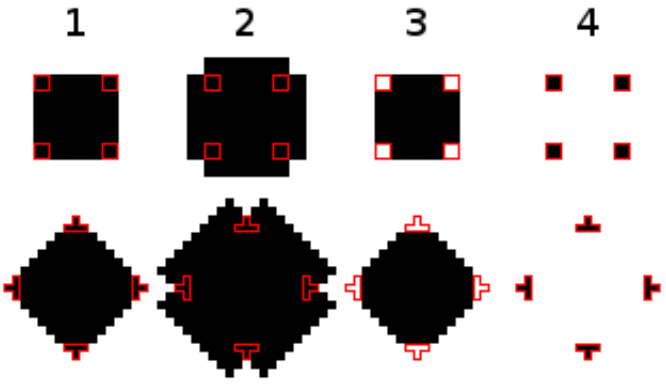

Figure 3.5. Dilate / Erode Corner Detection Steps

While the original author of this morphological corner detection algorithm is unknown, it is discussed by Robert Laganire in *OpenCV 2 Computer Vision Application Programming*

Cookbook [14]. Two stages of dilate and erode with different kernels are applied. The first stage uses a 5x5 "cross" dilate kernel to expand only horizontally and vertically. It then uses a 5x5 "diamond" erode kernel to shrink diagonally. The resulting image is compared with the original and those pixels which are in the corner of an aligned rectangle are found. The second stage uses a 5x5 "X" dilate kernel to expand in the two diagonal directions. It then uses a 5x5 "square" erode kernel to shrink horizontally and vertically. The resulting image is compared with the original and those pixels which are in a 45 degree corner are found. The resulting corners from both steps are combined into a final set of corners which are detected.

Figure 3.5 demonstrates these two stages. The top set of images corresponds to stage one with the "cross" and "diamond" kernels used to detect aligned corners. The bottom set of images corresponds to stage two with the "X" and "square" kernels used to detect 45 degree corners. Step one shows the original input of each stage, step two is the image after dilation, step three is the image after erosion, and step for is the difference of the original and eroded versions. The resulting corners are outlined in red in each step to provide a basis of how the dilate and erode operations modify the input.

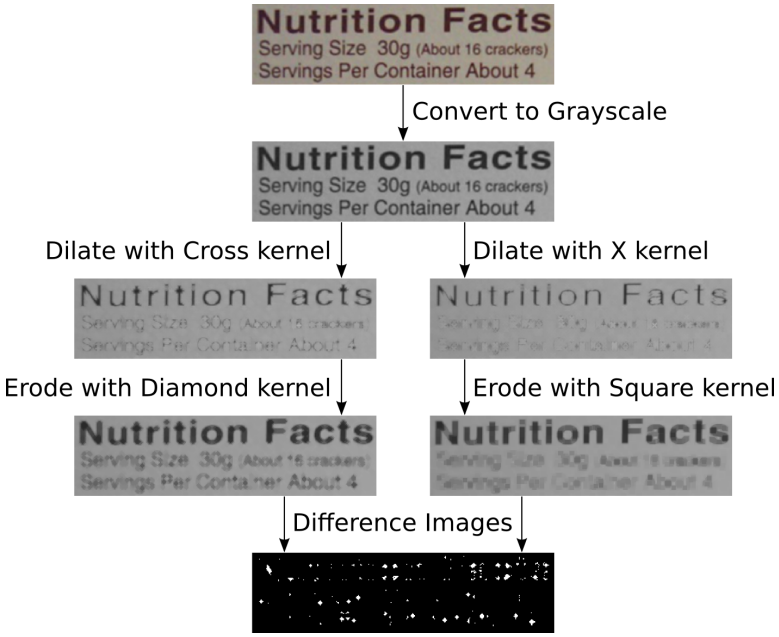

Figure 3.6. Dilate / Erode Corner Detection Example

Using a 5x5 kernel for all operations means that the dilate and erode operations generally expand and contract pixels by two at a time. In Figure 3.5 we see that the two types of detected corners contain more than just the most obvious corner pixel but that is because these examples are perfectly aligned. For the more common corners which are not perfectly aligned, fewer corner pixels are detected.

Figure 3.6 demonstrates this algorithm on an actual input. The dilate steps are substantially "whiter" than previous because the appropriate kernel has been used to expand white pixels. Then the erode steps partially reverse this brightening by expanding darker pixels. The result is still not like the original but the difference of the two is where corners have been detected.

CHAPTER 4
NUTRITION FACTS TABLE DETECTION AND LOCALIZATION

4.1 Introduction

While research has taken place to perform Nutrition Facts Table (NFT) Localization on images where it is known that such a table exists and is aligned properly, these conditions are actually quite rare when applied in real-world situations. Additionally, current solutions tend to perform very slowly on the mobile devices for which they are intended for use. A new approach to the problem will be presented which can both deal with less-ideal inputs and boast improved speed all while maintaining the same or better level of quality.

The problem definition now includes the ability to detect the presence of a NFT rather than assume that one is always in view; this is essential to improving overall speed as the algorithm can "fail fast" and skip to the next input instead of continuing to perform subsequent operations such as row detection, word splitting, and optical character recognition. Non-aligned inputs are also addressed to some extent by performing an intelligent rotation of the entire image before fully testing for the presence of the NFT.

Although this new method already has several benefits over previous ones, it actually opens doors towards even more reliable and extensible localization of NFTs by normalizing data in such a way as to greatly aid future analysis. It is very likely that further research can ultimately produce an algorithm that works with inputs of all rotations and continue to improve NFT localization precision while maintaining a high degree of specificity.

4.2 Goals and Considerations

The goals and considerations of this particular problem have played a large role in the

solution this paper presents. In particular we will discuss the exact problem definition, platforms and devices which were targeted, integration of existing libraries, input requirements, camera configurations.

The exact problem which we set out to solve consists of two parts: Does a given input image contain a nearly aligned NFT? If so, within which aligned rectangular area can the NFT be localized. For our concerns, a nearly aligned NFT is one which has been rotated away from perfect alignment by up to 30 degrees in either direction. Additionally, the entire input image may be rotated to align the NFT before the rectangular area specifying the NFT localization is computed.

We have targeted medium- to high-end Android devices which can consist of as little as a single core ARM System on a Chip with 1 GB RAM up to newer quad-core ARM SoC devices with 2 GB or more RAM. Due to the constrained hardware conditions which may be encountered, it is essential that this method use simple, off-the-shelf analysis methods to gather information about the input but combine their results in such a way as to produce reliable results. A related goal is to decrease processing time for each input from around 5 seconds to about 1 second.

One last consideration we must clarify is that of input quality. This method relies heavily upon detected edges and corners which requires that those areas of the input image which contain the NFT to be detected are clearly captured by the device camera. Conditions to produce a quality input image include camera focus, area lighting conditions, and overall stability. Certain steps can and have been made on those devices which support such configurations. These include a request that the center of the image always be kept in focus and perform a faster search for focus when needed. Figure 4.1 exemplifies the need for centered camera focus: The left image is focused on the region behind the NFT which produces very poor quality input for analysis while the right image is properly focused on the NFT and is therefore a high quality input image.

4.3 Early Rotation Correction

Before any real localization work can begin, a rotation correction step is performed

Figure 4.1. Camera Focus versus Input Image Quality

to align inputs which may only be nearly aligned. This correction is performed by taking advantage of the high level of regular horizontal lines found within a NFT. All detected lines which are horizontal within 30 degrees in either direction are used to compute an average horizontal rotation which is then used to perform the appropriate correcting rotation. An example of this can be found in Figure 4.2 where the left image is the raw input from the device and the right image has been corrected for rotation. Although the actual rotation in this example is quite low at approximately 3 degrees counterclockwise, note the black corners which are introduced by the rotation itself.

The exact steps undertaken to perform this correction include a conversion from the colored input image to a grayscale copy upon which the Canny Edge Detector can be applied. Edge detection results are subsequently fed into a Hough Transformation which provides a list of all detected lines in the image. All lines are analyzed to determine if they are within 30 degrees of a horizontal line and θ values are averaged together for those that pass this test. The final averaged θ value is used in reverse to rotate the image in such a way as to make the new average horizontal rotation much closer to perfectly horizontal.

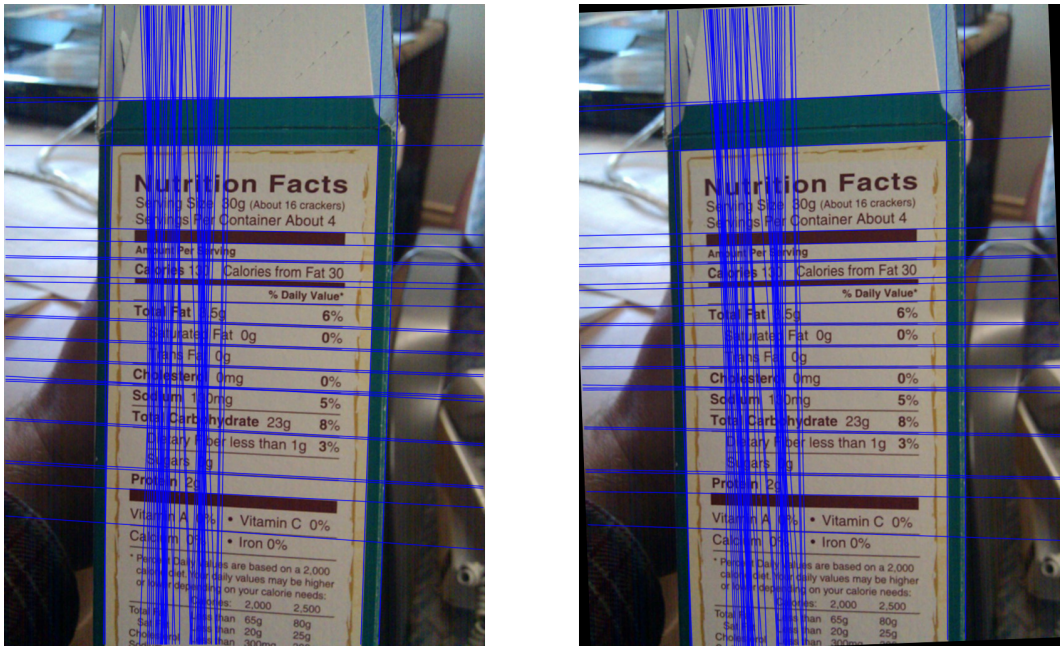

Figure 4.2. Example Rotation Correction

These steps are summarized in Algorithm 1.

4.4 Corner Detection and Analysis

Now the rotation corrected image is ready for corner detection. The Dilate/Erode Corner Detector is applied to retrieve a two-dimensional bitmap where "true" white pixels correspond to detected corners and all other "false" pixels are black. This image on its own is not very useful but two very useful projections can be produced from it.

The projections themselves are simply a sum of the "true" pixels for each row and column. Figure 4.4 demonstrates four of these sums, two rows and two columns. The green and teal row projections sum to 0 and 12, respectively. The red and orange column projections sum to 0 and 10, respectively. Similar sums are performed for every other row and column which produce two arrays of the sums of row and column projections. The row projection will have an entry for each row in the image while the column projection will have an entry for each column in the image.

The purpose of these projections is to determine "edges" for the top, bottom, left, and

Algorithm 1: Early Rotation Correction

input : possiblyRotatedImage
output: rotationCorrectedImage

```
1  grayscaleImage ← ConvertToGrayscale(possiblyRotatedImage);
2  detectedEdges ← CannyEdgeDetector(grayscaleImage);
3  detectedLines ← HoughTransformation(detectedEdges);
4  lineCount ← 0;
5  thetaTotal ← 0;
6  for ρ, θ in detectedLines do
7      if θ is within 30 degrees of horizontal then
8          lineCount ← lineCount + 1;
9          thetaTotal ← thetaTotal + θ;
10     end
11 end
12 thetaAverage ← Round(thetaTotal / lineCount);
13 if thetaAverage ≠ 0 then
14     rotationCorrectedImage ← RotateImage(possiblyRotatedImage, -thetaAverage);
15 else
16     rotationCorrectedImage ← possiblyRotatedImage;
17 end
```

right sides of the region in which most corners lie. The Dilate/Erode Corner Detector is used specifically because of its high sensitivity to contrasted text so we assume that the region is bounded by these edges contains a large amount of text. Areas of the input image which are not in focus do not produce a large amount of corner detection results and so will tend to not lie within the projection edges we seek.

Calculation of projection edges is quite simple after the projection has been built. Each value of the projection is averaged together and we select a projection threshold equal to twice the average. Once a projection threshold is selected, the first and last indexes of each projection which pass the threshold are selected as the edges of that projection. Figure 4.5 is a close-up view of the column projection from Figure 4.3 where projection values are colored green where they are not over the threshold and teal where they are over the threshold. The first and last columns with passing teal values have been marked with red lines and would be used as the edges of this particular corner projection.

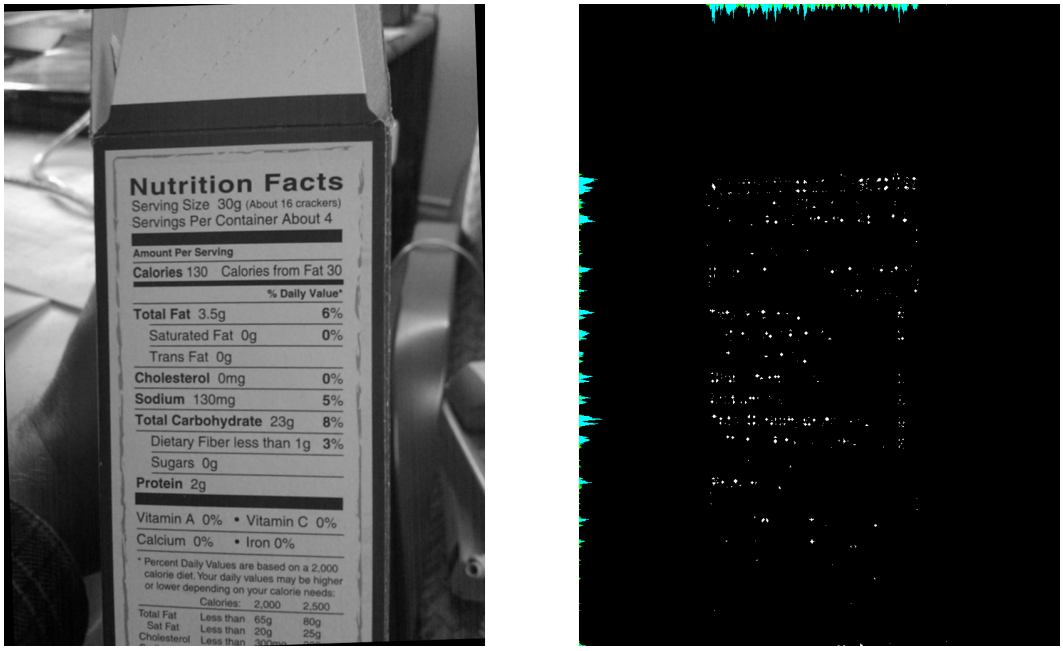

Figure 4.3. Example Corner Detection and Projections

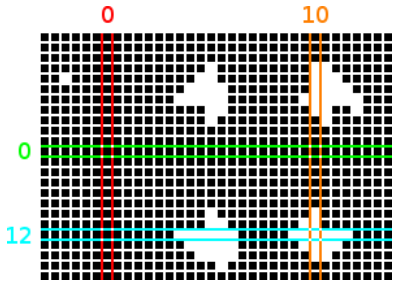

Figure 4.4. Corner Projection Explanation

Algorithm 2 covers the process taken to perform corner detection and projection followed by projection edge selection. Lines 1 - 6 perform setup and actual corner detection by calling into the Dilate / Erode Edge Detection image analysis algorithm. Lines 7 - 16

Figure 4.5. Example Corner Projection Edge Selection

handle the summation of detected corners into both row and column projections by iterating through every row and column combination and incrementing the appropriate values of the row and column projection counters when a detected corner is found. Lines 17 - 28 find the top and bottom edges of the row projection by determining the threshold and iterating over each value of the row projection to find the top and bottom locations which pass the threshold. Lines 29 - 40 find the left and right edges of the column projection by determining the threshold and iterating over each value of the column projection to find the left and right locations which pass the threshold. It should be also noted that if any of the projection edges can not be determined, this method will immediately classify the image as not containing any NFT.

It is of interest to point out the distinct patterns commonly found in these two projections: The row projection tends to create evenly spaced short spikes for text in each line of text within the NFT while the column projection tends to contain one very large spike where the NFT begins at the left due to the sudden influx of text which was detected. No in-depth analysis of these patterns was performed in this research but projection data was collected for each image tested and it is expected that research in this area will allow for detection and corresponding correction of inputs with all rotations. The column projections could be used for greater accuracy in determining the left and right bounds of the NFT while row projections could be used by later analysis steps such as row division. Additionally, certain projection "profiles" could eventually be used to select a specifically tailored approach to localization.

4.5 Selection of Bounding Hough Lines

Now that all four projection edges have been determined, the next step is to select those Hough lines which fit closest to the edges without crossing into them. Figure 4.6 provides an example of the state before and after this step. Although a Hough Transformation was performed in during the early rotation correction in Section 4.3, the image could have been rotated so they must be calculated once again. The grayscale image which was used with the Dilate/Erode Corner Detector in Section 4.4 is reused with the Canny Edge Detector

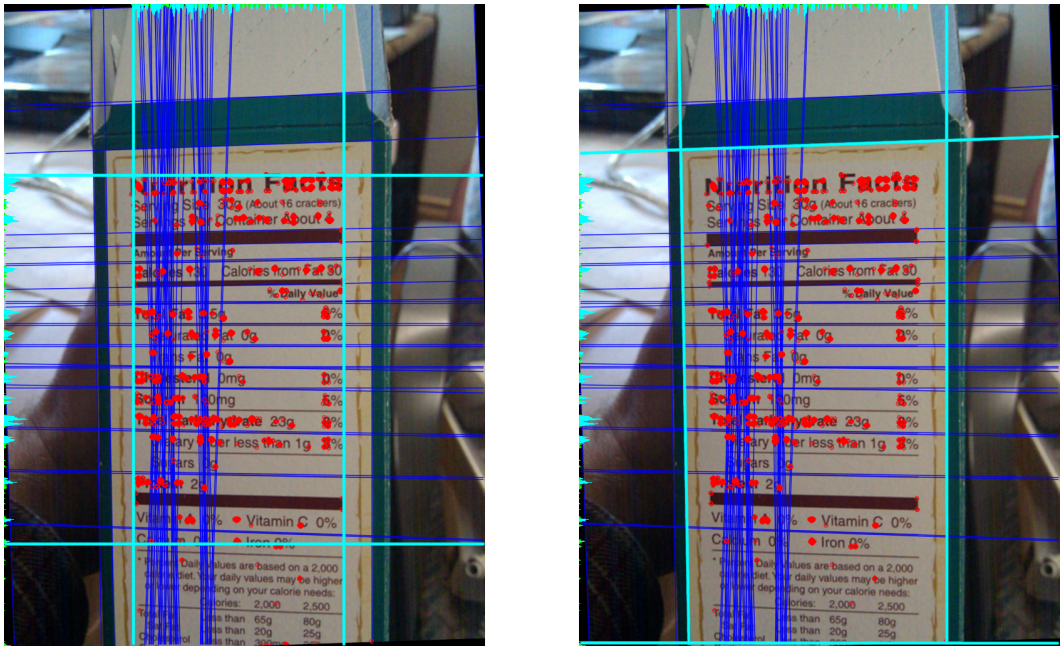

Figure 4.6. Example Boundary Selection with Corner Projection Edges and Hough Lines

and detected edges are analyzed with another Hough Transformation. During the early rotation correction of Section 4.3, only nearly horizontal lines were of importance but this time both horizontal and vertical lines are needed. In addition, a stricter threshold of 5 degrees is placed on their rotations because the image is expected to already be aligned. Every line returned from the Hough Transformation is tested and stored as being either horizontal or vertical if it is within this threshold.

Algorithm 3 demonstrates the exact steps which are followed for this phase. Lines 1 - 4 cover setup, edge detection, and Hough Transformation. Lines 5 - 11 perform strict horizontal and vertical tests on the Hough Transformation results to determine which are horizontal or vertical. Lines 12 - 25 and 26 - 39 perform the vertical boundary and horizontal boundary selection, respectively. Lines 12, 13, 26, and 27 set the default starting boundary lines at the four sides of the image. Lines 14 - 25 test each of the saved vertical lines to find the best-fitting top and bottom boundaries. Lines 28 - 39 test each of the saved horizontal lines to find the best-fitting left and right boundaries.

The exact test to determine if a given Hough line involves comparing the values where

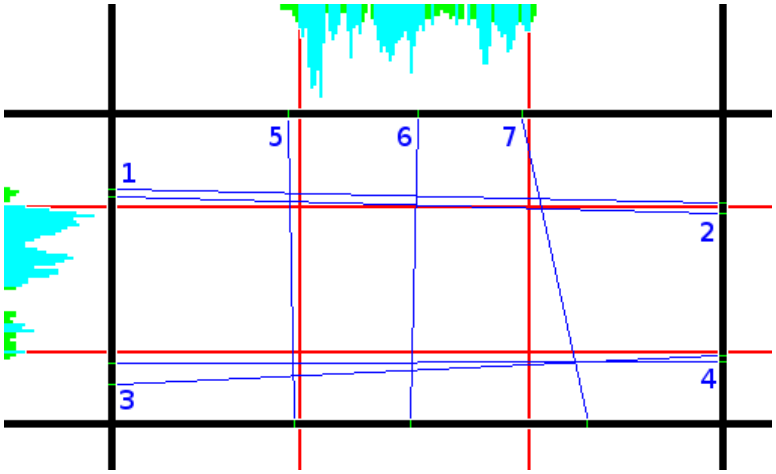

Figure 4.7. Boundary Selection Examples

it intersects the outside edges of the image. For example, a horizontal Hough line intersects the left and right edges of the image at two possibly different rows and a vertical Hough line intersects the top and bottom edges of the image at two possibly different columns. Figure 4.7 shows these intersections with green marks where the 7 Hough lines meet the outside of the image area. These intersection values are first tested against the appropriate projection edges to determine if the line lies entire outside of the projection edges or if it enters the region even partially. Figure 4.7 marks the projection edges with red lines within the image. Those lines are entirely outside the projection edges and have either the minimum or maximum sum of intersection values are kept as the current best boundary lines. Lines 2, 6, and 7 in Figure 4.7 will immediately be excluded because they all have at least one image intersection inside the projection edges. Lines 1 and 5 are valid boundaries with no others to compete with. Lines 3 and 4 could both be valid bottom boundary lines but 4 will be selected over 3 because it has a lower sum of image intersection values. This test can be found on lines 16, 20, 30, and 34 of Algorithm 3 where left, right, top, and bottom edges are tested, respectively.

It is very possible that appropriate boundary lines cannot be found in each of the four locations. This usually occurs when the NFT itself is cut-off by the edge of the input image but can also be caused by a lack of detected edges and therefore Hough lines in blurry areas

of the input image. These missing boundary lines can usually be satisfied with the starting default boundaries at the far edges of the image; however, if too many of the boundary lines must be defaulted to their edge it is highly likely that the results are just not reliable enough and there is not any NFT to be found. We have found that a boundary threshold of 2 works well for most situations - especially because it is not uncommon for a NFT to be cut-off at both the top and bottom edges of an input image. If fewer than two Hough boundary lines are found at this stage then the method immediately classifies the input as not containing any NFT.

4.6 Simplification of Bounded Area

It is highly likely that the bounded area will not be perfectly rectangular which makes integration with later analysis where an aligned rectangular area is expected very difficult. This is circumvented by fitting an aligned rectangle around the selected Hough boundary lines but each of the four boundary intersections must be found in order to do so. It is also likely that any further analysis which accepted a four-sided polygon would also rather receive four pixel coordinates of the boundary intersections than four (ρ, θ) pairs.

While computing the intersection of two linear equations in Cartesian coordinate space is a simple problem, performing the same calculation within polar coordinate space line is not nearly as straightforward. Chapter 5 discusses this matter in great detail so we will gloss over it for now and just say that there is a simple function that takes two polar coordinate lines and returns the pixel coordinate of their intersection.

As seen in Figure 4.8, after the four intersection coordinates are obtained their components can be compared and combined to find the minimum aligned rectangle which fits the entire bounded area. We see that left image has four orange points at the four intersections and then the right image displays the simplified boundary which contains all four points. This aligned rectangle is the final result of this NFT localization algorithm and can be passed on to other algorithms which perform further analysis such as row dividing, word splitting, and optical character recognition.

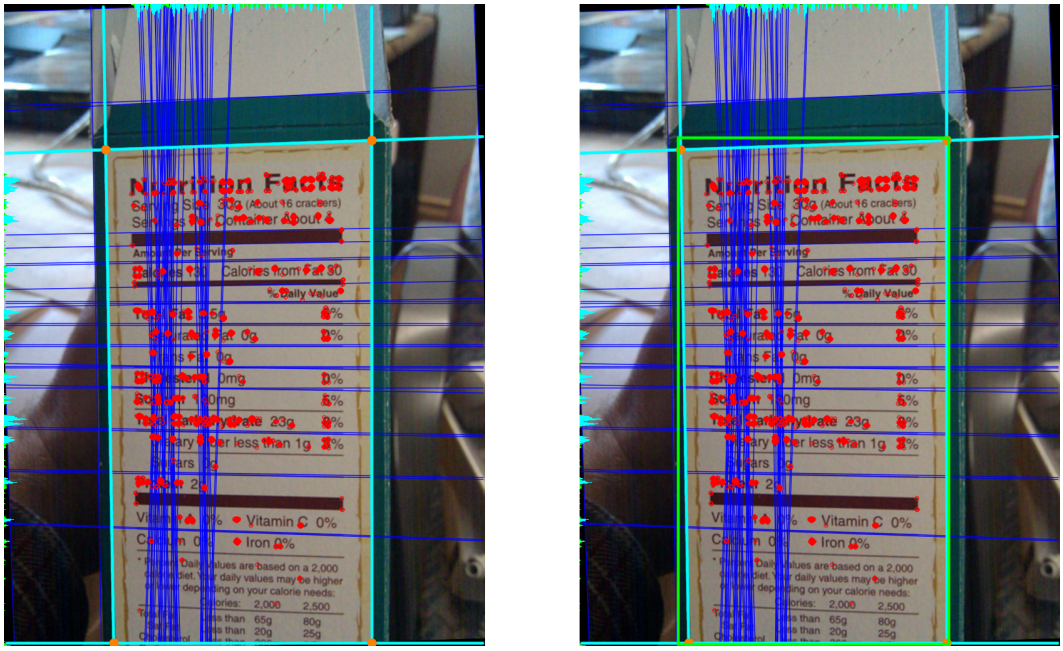

Figure 4.8. Simplification of Hough Boundary Lines

4.7 Review

A flow chart documenting this method can be found in Figure 4.9. States *Start*, *a*, *b*, *c*, and *d* correspond to the steps taken during the "Early Rotation Correction" described in Section 4.3. States *d*, *e*, *h*, *i* correspond to the steps taken during "Corner Detection and Analysis" described in Section 4.4. States *d*, *e*, *f*, *g* and *j* correspond to the steps taken during "Selection of Bounding Hough Lines" described in Section 4.5. States *j* and *Finish* correspond to the steps taken during "Simplification of Bounded Area" described in Section 4.6.

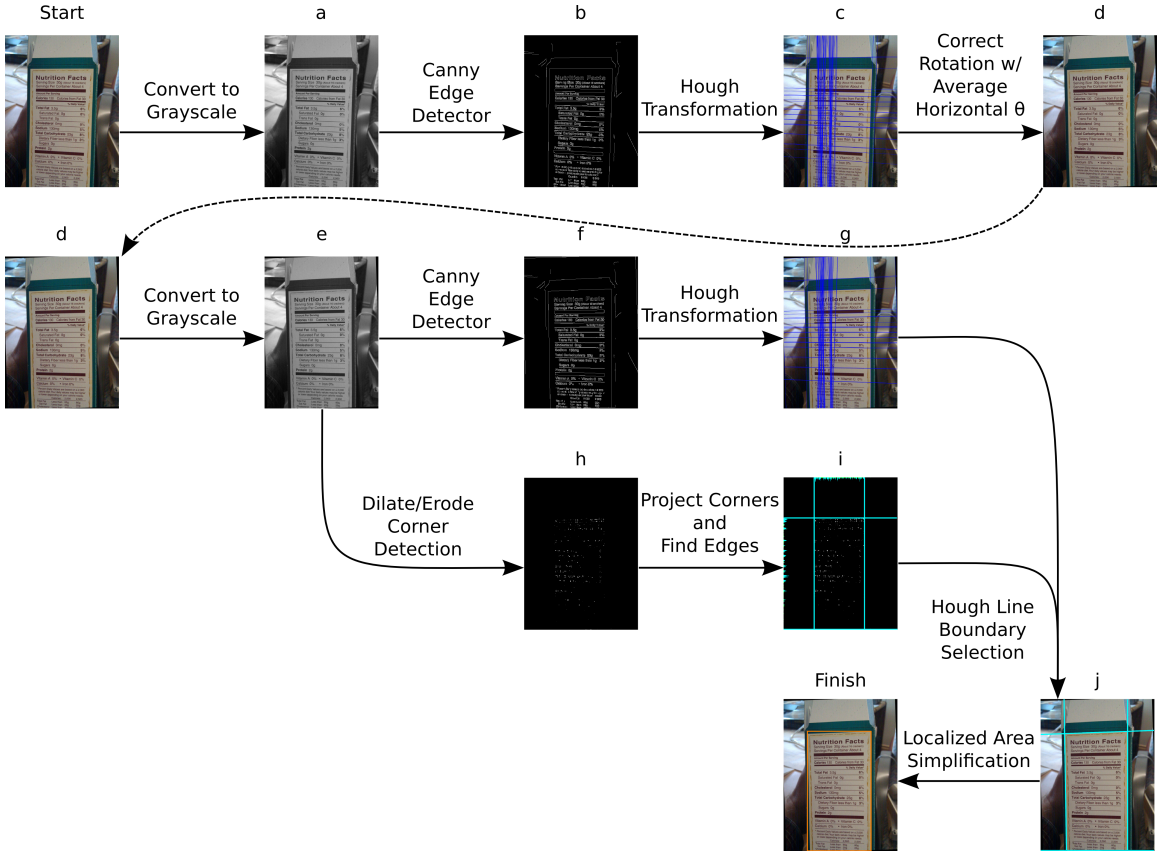

Figure 4.9. Process Flow Chart

Algorithm 2: Corner Detection, Projection, and Edge Analysis

input : rotationCorrectedImage
output: topEdge, bottomEdge, leftEdge, and rightEdge projection edges

```
rows ← number of rows in rotationCorrectedImage;
cols ← number of columns in rotationCorrectedImage;
rowProjection ← array of size rows;
colProjection ← array of size cols;
grayscaleImage ← ConvertToGrayscale(rotationCorrectedImage);
detectedCorners ← DilateErodeCornerDetector(grayscaleImage);
totalCorners ← 0;
for row in (0, rows] do
    for col in (0, cols] do
        if detectedCorners_{col,row} is white then
            rowProjection_{row} ← rowProjection_{row} + 1;
            colProjection_{col} ← colProjection_{col} + 1;
            totalCorners ← totalCorners + 1;
        end
    end
end
rowThreshold ← totalCorners × 2 ÷ rows;
topEdge ← undefined;
bottomEdge ← undefined;
for row in (0, rows] do
    if rowProjection_{row} > rowThreshold then
        if topEdge is undefined then
            topEdge ← row;
        else
            bottomEdge ← row;
        end
    end
end
colThreshold ← totalCorners × 2 ÷ cols;
leftEdge ← undefined;
rightEdge ← undefined;
for col in (0, cols] do
    if colProjection_{col} > colThreshold then
        if leftEdge is undefined then
            leftEdge ← row;
        else
            rightEdge ← row;
        end
    end
end
```

Algorithm 3: Determination of Hough Boundary Lines

input : grayscaleImage, topEdge, bottomEdge, leftEdge, and rightEdge
output: topLine, bottomLine, leftLine, and rightLine Hough boundary lines

```
rows ← number of rows in grayscaleImage;
cols ← number of columns in grayscaleImage;
detectedEdges ← CannyEdgeDetector(grayscaleImage);
detectedLines ← HoughTransformation(detectedEdges);
for ρ, θ in detectedLines do
    if θ is within 5 degrees of horizontal then
        add ρ, θ to horizontalLines;
    else if ρ is within 5 degrees of vertical then
        add ρ, θ to verticalLines;
    end
end
topLine ← 0, π/2;
bottomLine ← rows, π/2;
for ρ, θ in verticalLines do
    topIntersect, bottomIntersect ← GetVerticalIntersects(ρ, θ);
    if topIntersect < leftEdge and bottomIntersect < leftEdge then
        if topIntersect + bottomIntersect > sum of intersects for leftLine then
            leftLine ← ρ, θ;
        end
    else if topIntersect > rightEdge and bottomIntersect > rightEdge then
        if topIntersect + bottomIntersect < sum of intersects for rightLine then
            rightLine ← ρ, θ;
        end
    end
end
leftLine ← 0, 0;
rightLine ← 0, cols;
for ρ, θ in horizontalLines do
    leftIntersect, rightIntersect ← GetHorizontalIntersects(ρ, θ);
    if leftIntersect < topEdge and rightIntersect < topEdge then
        if leftIntersect + rightIntersect > sum of intersects for topLine then
            topLine ← ρ, θ;
        end
    else if leftIntersect > bottomEdge and rightIntersect > bottomEdge then
        if leftIntersect + rightIntersect < sum of intersects for bottomLine then
            bottomLine ← ρ, θ;
        end
    end
end
```

Algorithm 4: Rectangular Fitting of Bounded Area

input : topLine, bottomLine, leftLine, and rightLine
output: topLeftCoord and bottomRightCoord

```
1 topLeftCoord ← SolveIntersection(topLine, leftLine);
2 topRightCoord ← SolveIntersection(topLine, rightLine);
3 bottomLeftCoord ← SolveIntersection(bottomLine, leftLine);
4 bottomRightCoord ← SolveIntersection(bottomLine, rightLine);
5 topLeftCoord_x ← Min(topLeftCoord_x, bottomLeftCoord_x);
6 topLeftCoord_y ← Min(topLeftCoord_y, topRightCoord_y);
7 bottomRightCoord_x ← Max(topRightCoord_x, bottomRightCoord_x);
8 bottomRightCoord_y ← Max(bottomLeftCoord_y, bottomRightCoord_y);
```

CHAPTER 5
SYSTEM OF LINEAR EQUATIONS IN POLAR COORDINATE SPACE

5.1 Explanation of Problem

After the selection of left, right, upper, and lower bounding Hough lines has taken place, the four intersections must be computed to determine the rectangular area which contains the entire bounded region. The exact method for this computation was previously skipped in lieu of a later explanation which this chapter will now address.

As previously stated, the exact pair of values which define each Hough line returned from a Hough Line Transformation are ρ and θ. ρ is the length of the normal which connects the line to the origin of polar coordinate space and θ is the angle about the origin which the initially vertical line is rotated. The use of (ρ, θ) pairs to define Hough lines is not accidental; this is the only way to clearly define vertical lines which must deal with the concept of an infinite slope in Cartesian coordinate space. However unlike solving systems of linear equations in Cartesian coordinate space, a fast and reliable method to find the Cartesian coordinate of the intersection of two polar coordinate lines is not a simple task.

Several approaches to solve this problem were taken with the final method using a purely trigonometric solution which will now be presented.

5.2 Solution Overview

We start with two lines defined in polar coordinate space: $l_1 = (\rho_1, \theta_1)$ and $l_2 = (\rho_2, \theta_2)$. First we must check for the two cases of no solution or all solutions by testing $\theta_1 = \theta_2$. If this is true and $\rho_1 = \rho_2$ then these are the exact same line and there is an infinite number of solutions. If the θ values are equal and $\rho_1 \neq \rho_2$ then these are different but parallel lines

and there are no solutions. In all other cases, we now start working with the two normals which define l_1 and l_2. We will delineate the line segments of these normals as n_1 and n_2. Line segment n_1 goes from the origin at Point A to l_1 at Point B while n_2 goes from Point A to l_2 at Point C. A new line segment l_3 can be drawn between points B and C to form triangle T_1 consisting of points A, B and C. Triangle T_1 will be used to form the basis of another triangle T_2 consisting of points B, C, and D where point D is the intersection of lines l_1 and l_2 we seek. Of the two new line segments in T_2, l_4 between points B and D and l_5 between points C and D, only l_4 need be found as the location of point D can be easily determined by adding line segments n_1 and l_4 as vectors.

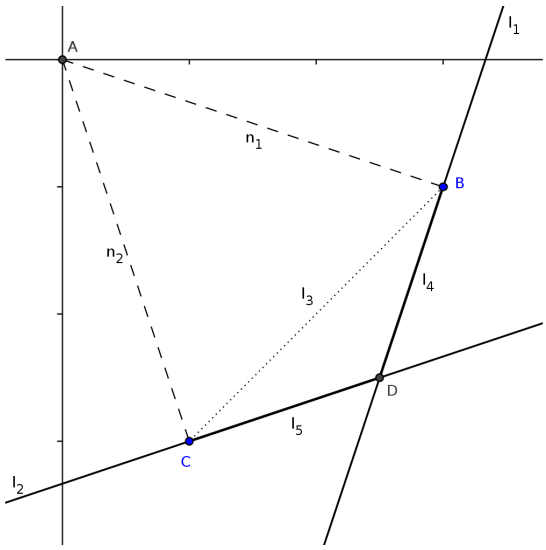

Figure 5.1. Case 1: Intersection lies between the two normals

There are three cases that must be dealt with in forming T_2 from T_1: the intersection lies within the region between the two normals shown in Figure 5.1, the intersection lines above the region between the two normals shown in Figure 5.2, or the intersection lies below the region between the two normals shown in Figure 5.3.

Once the location of the intersection is narrowed down to one of these three cases, T_2 can be built in the direction of the intersection. More specifically, in case 1 where the the intersection is between the two normals, the portion of l_1 below n_1 will become a new line segment l_4 and the portion of l_2 above n_2 will become a new line segment l_5. Both l_4 and

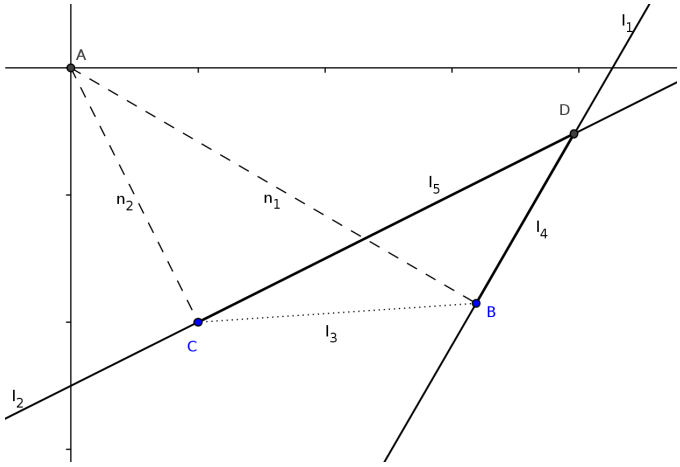

Figure 5.2. Case 2: Intersection lies above the two normals

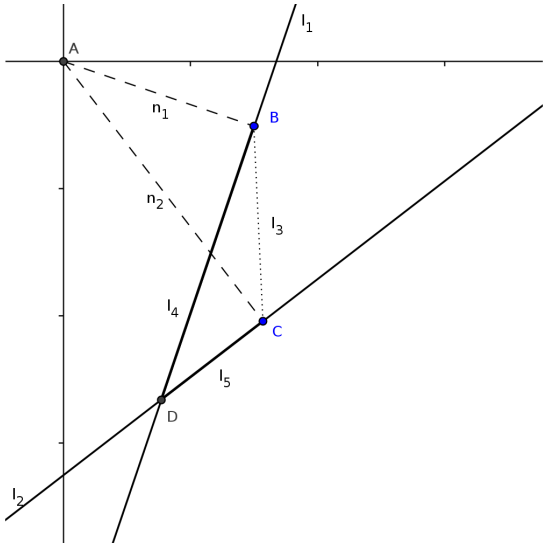

Figure 5.3. Case 3: Intersection lies below the two normals

l_5 terminate at point D and between the length of l_3 and the angles BCD and CBD there is enough information to solve for the length of l_4 with the Law of Cosines. This logic is similar in the other two cases except for one of the two new line segments would extend above or below both normals in cases 2 and 3, respectively.

5.3 Precise Solution

Let (ρ_1, θ_1) and (ρ_2, θ_2) define the two input lines where $\rho_1 \leq \rho_2$. Now fill out the details of T_1:

$$\theta_3 = \theta_2 - \theta_1 \tag{5.1}$$

$$\rho_3 = \sqrt{\rho_1^2 + \rho_2^2 - 2\rho_1\rho_2 \cos\theta_3} \tag{5.2}$$

$$\theta_4 = \arcsin(\frac{\rho_2}{\rho_3} \sin\theta_3) \tag{5.3}$$

$$\theta_5 = \pi - \theta_3 - \theta_4 \tag{5.4}$$

The angle between the two normals is found in Equation 5.1 which we know will be positive because $theta_2 >= theta_1$. The Law of Cosines is used in Equation 5.2 to get the distance of the line segment between the two normals. The next two equations find the remaining inner angles of T_1. Values of θ_6 and θ_7 for the most common case 1 are the remaining angles between θ_4 and θ_5 and their respective perpendiculars:

$$\theta_6 = \frac{\pi}{2} - \theta_4 \tag{5.5}$$

$$\theta_7 = \frac{\pi}{2} - \theta_5 \tag{5.6}$$

However if a test for case 2 must be made to see if l_2 intersects n_1: If $\rho_1 > \rho_2$ then test for $\rho_1' < \rho_1$ where $\rho_1' = \frac{\rho_2}{\cos\theta_3}$. In the case that this is true, we redefine $\theta_6 = \theta_4 + \frac{\pi}{2}$ and $\theta_7 = \theta_5 - \frac{\pi}{2}$. It is only in this case that we need to reverse the projection from l_1 since it projects above the normal rather than below.

Also a test for case 3 should be performed by checking if l_1 intersects n_2: If $\rho_1 < \rho_2$ then test for $\rho_2' < \rho_2$ where $\rho_2' = \frac{\rho_1}{\cos\theta_3}$. In the case that this is true, we redefine $\theta_6 = \theta_4 - \frac{\pi}{2}$ and $\theta_7 = \theta_5 + \frac{\pi}{2}$.

After tests for the three cases are performed, we need only determine the length of the line segment l_4:

$$\theta_8 = \pi - \theta_6 - \theta_7 \tag{5.7}$$

$$\rho_5 = \rho_3 \frac{\sin\theta_7}{\sin\theta_8} \tag{5.8}$$

Earlier we mentioned that the projection from l_1 must be reversed for case 2. Normally $\theta_9 = \theta_1 + \frac{\pi}{2}$ but in the case of a reversal we define $\theta_9 = \theta_1 - \frac{\pi}{2}$. Now there is enough information to decompose n_1 and l_4 and combine them to determine the exact coordinates of the intersection of l_1 and l_2:

$$(\rho_1 \cos\theta_1 + \rho_5 \cos\theta_9, \rho_1 \sin\theta_1 + \rho_5 \sin\theta_9) \tag{5.9}$$

CHAPTER 6

TESTING

6.1 Procedure

We assembled 378 images taken with a mobile device during a typical shopping session at a grocery store. Of these images, 266 contain a NFT and 112 do not contain any NFT. These images were tested on a mobile device with the same analysis code that is meant for real-time usage. This method is only capable of determining the existence of a NFT in a given input and returning a rectangular area for those inputs where it believes a NFT exists so we manually sorted the results into five categories: complete true positive, partial true positive, true negative, false negative, and false positive.

It should be noted that no exact calculations were performed as a threshold between complete true positive and partial true positive; only a general guideline that the entire NFT along with minimal non-NFT area be matched for a complete true positive while a partial true positive matches at least some portion of the NFT or other non-NFT areas which may cause interference during later analysis. This general guideline is explained quite thoroughly in Section 6.2.

It should be noted that this method is meant to be run on a steady stream of input images. It is reasonable to have a lower match rate for inputs which contain a NFT because there are several opportunities to find a match and only one of many inputs needs to succeed. On the other hand, detecting a NFT within an image which does not contain any NFT is unacceptable because it will waste system resources and produce incorrect and confusing results for the user. In more technical terms, the application of this method requires high specificity with the trade-off of low recall.

6.2 Distinguishing Between Complete and Partial True Positives

The general questions we used in determining if a given true positive result is complete or partial are as follows:

- Does the localized region contain the entirety of the NFT?
- Does the localized region not contain areas to the sides of the NFT which could interfere with further analysis such as in-focus text or graphics?
- Is the localized region mostly aligned with the NFT?

If each question can be answered "yes" then it is highly likely that a complete match is made. If any are answered "no" it becomes more probable that only a partial match is made. Figure 6.1 displays a standard a complete and partial match which can be easily determined using these rules. Of course, the full scope of this issue is not quite this simple and we had a handful of results which were still difficult to classify with these questions.

Figure 6.1. Example Complete and Partial True Positive Results

Figure 6.2 displays another complete and partial match from left to right, respectively. Although quite similar, there are still definite properties which can be used to make the decision. The complete match on the left is missing a partial amount of the right edge of the NFT but this is a corner-case which shows up in enough results to warrant a simple padding of the right side of the localized region. Additionally, it contains a small amount of non-NFT area to the left but there are no details to speak of within this region so further analysis will not be negatively impacted. Larger non-NFT regions lie above and below the localized region but further analysis is expected to reject these regions after performing some kind of row detection analysis. The partial match on the right does contain most of the NFT but essential text has been excluded from the left side of the NFT and this is not a common enough scenario to allow for an exception. Both results are slightly rotated but not enough to single-handedly exclude either of them from being a complete match.

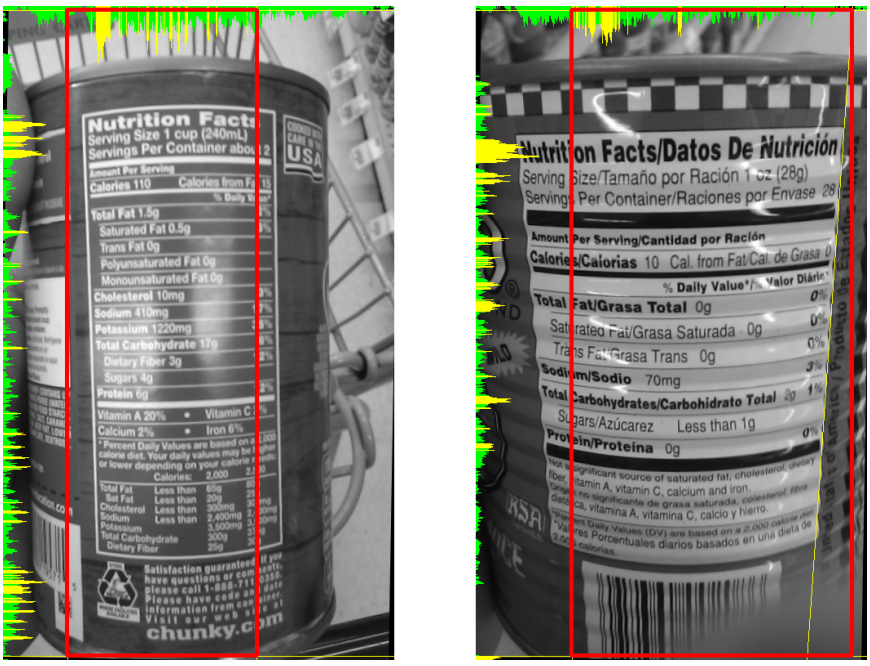

Figure 6.2. Difficult Categorizations

Figure 6.3 is one last example of difficult complete and partial matches from left to right, respectively. The complete true positive on the left does contain some text, specifi-

cally an area of the ingredients list, which could be confusing for further analysis but the NFT is closely matched with the upper left corner and has enough detail to aid in a proper analysis without debilitating errors. Meanwhile, the partial match on the right is an example of the difficult variety of NFT layouts which can be encountered in real-world use. While this method was able to correctly discern the presence of a NFT, the localized region is far to large to be of any use and is missing essential text on the left of the NFT.

Figure 6.3. Other Difficult Categorizations

6.3 Results

Of the 266 images tested which contain some kind of NFT, 83 were matched as a complete true positive and 27 were matched as a partial true positive. This gives a total true positive match rate of 42% and associated false negative match rate of 58%. All test images which do not contain any NFT were properly matched as a true negative.

Further analysis of these results can be conveyed via precision, recall, specificity, and accuracy. These terms are defined by Douglas Altman and Martin Bland in *Diagnostic*

tests. 1: Sensitivity and specificity but a quick explanation follows [1]. Precision is the percentage of complete true positive matches out of all true positive matches. Recall is the percentage of true positive matches out of all possible positive matches. Specificity is the percentage of true negative matches out of all possible negative matches. Accuracy is the percentage of true matches out of all possible matches. More specifically, precision, recall, specificity, and accuracy can be mathematically defined given the five types of matches that we categorized results into: complete true positive ($TP_{complete}$), partial true positive ($TP_{partial}$), true negative (TN), false positive (FP), and false negative (FN).

$$Precision = \frac{TP_{complete}}{TP_{complete} + TP_{partial}} \tag{6.1}$$

$$Recall = \frac{TP_{complete} + TP_{partial}}{TP_{complete} + TP_{partial} + FN} \tag{6.2}$$

$$Specificity = \frac{TN}{TN + FP} \tag{6.3}$$

$$Accuracy = \frac{TP_{complete} + TP_{partial} + TN}{TP_{complete} + TP_{partial} + TN + FP + FN} \tag{6.4}$$

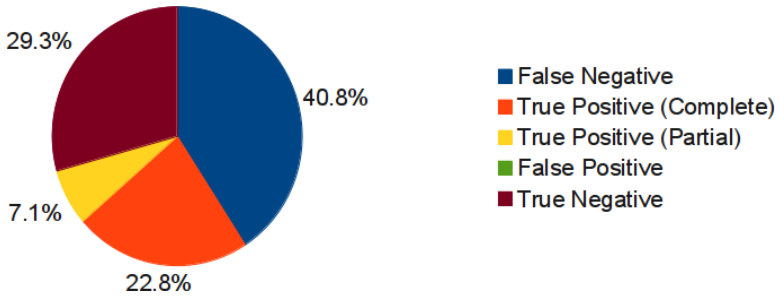

Figure 6.4. NFT Localization Results

Precision	Total Recall	Complete Recall	Partial Recall	Specificity	Accuracy
0.7632	0.4222	0.3580	0.1475	1.0	0.5916

Table 6.1. NFT Localization Performance Data

Table 6.1 shows the calculated precision, recall, specificity and accuracy for these results. Recall from Section 6.1 that the application for this algorithm requires high specificity

with the trade-off of low recall so these first results look very promising. Real-world testing has not been extensively performed yet and we expect to see the specificity lower as more varied inputs are analyzed. Unfortunately, total and especially complete recall are somewhat low. This may not matter very much because of the fast rate at which input images can be processed on target devices, but there is definitely room for improvement.

6.4 Limitations

6.4.1 Blurry Inputs

The limitation shown in Figure 6.5 which causes a majority of false negative matches is inability to use blurry inputs which are generally caused by a combination of camera focus and stability. Both the Canny Edge Detector and Dilate/Erode Corner Detector require rapid and contrasting changes to identify key points and lines of interest. These points and lines are meant to correspond directly with text and NFT borders so when useful data can't be retrieved from a blurry image the analysis gives up quickly.

Our only recourse to deal with blurry inputs is improved camera focusing and stability; both of which are mostly outside the scope of this method. As mentioned in Section 4.2, the current Android application does attempt to force focus within at the image center but this ability is not present in older versions. Over time, as device cameras improve and more devices run newer versions of Android, this limitation will have less impact on recall but it will never be fixed entirely.

6.4.2 Non-Square NFTs

It goes without saying but a large amount of grocery packaging does not use flat cardboard boxes. One such example can be seen in Figure 6.6. Since this method uses a combination of detected corners and lines to perform its analysis, not much can happen if no straight bounding lines are found. Bottles, bags, cans, and jars all have a large showing in the false negative category because of their lack of detected Hough lines during analysis.

One possibility to get around this limitation is a more rigorous line detection step in

Figure 6.5. Example Blurry Input

which a Segmented Hough Transform (see Subsection 7.2.7) is performed and regions which contain connecting detected lines are grouped together. These grouped regions could be used to warp a curved image into a rectangular area for further analysis.

6.4.3 Irregular and Busy Inputs

Smaller grocery packages like those in Figure 6.7 tend to place a large amount of information into a very tiny space. In these situations it is also quite common to organize NFT information with irregular layouts. These complicated inputs combined present an extremely difficult problem for analysis. Our method is specifically tailored for more traditional stacked NFT layouts with generally empty surrounding areas and it makes certain simplifying assumptions to target these more common NFTs.

As better analysis of corner projections and Hough lines is integrated into this method,

Figure 6.6. Example Curved Input

it will become possible to classify inputs as definitely traditional or more irregular and/or busy. If this classification can work reliably, the method could switch to a much slower and generalized localization to produce better results in this situation while still quickly returning results for the more common layouts.

Figure 6.7. Example Irregular and Busy Inputs

CHAPTER 7

CONCLUSION

7.1 Overview

Although this is just the first step towards extracting useful data from NFTs, this method performs an integral part of the process and has been designed to cleanly fit into place with other components in active research.

While work progressed on this method, we took note of many possible improvements which were out of scope at the time. The following section documents these future work items however it should be noted that the changes required to implement some of them may preclude the possibility of others.

7.2 Future Work

7.2.1 Improved Canny Edge Detector Thresholds

The current static thresholds used within the Canny Edge Detector work well in most situations but can provide too little or too much detail in strange lighting conditions. Since this method relies on a steady stream of inputs, an automated threshold adjustment based on previous Canny Edge Detector results could allow for better results over time in these situations. Alternatively, an improved version of the Canny Edge Detector which manages its own thresholds could be tested [7].

7.2.2 Custom Hough Transform Algorithm

Because input images are generally not square, the Hough Transform returns more results for lines in the longer dimension because they are more likely to pass the threshold.

Being able to specify different thresholds for the two dimensions and intelligently combining them for various rotations would produce more consistent results.

Since only those Hough lines which are nearly vertical or horizontal are of use to this method, improvements can be made by only allocating "bins" for those ρ and θ combinations that have importance. Fewer bins means less memory to track all of them and fewer tests to determine which bins need to be incremented for a given input.

7.2.3 Faster Corner Detection Replacement

There are many more efficient corner detection algorithms but not all of them will produce results like the current Dilate / Erode Corner Detector does. An analysis of various replacements could be performed to see if another algorithm can be used which won't alter overall results but still provide speed and/or memory improvements.

7.2.4 Better Auto-correction of Rotated Inputs

More intelligent analysis of the resulting Hough lines during early rotate correction could allow for properly detecting and localizing NFTs in inputs of all rotations. Searching for Hough lines with similar θ values and even spacing of ρ values could provide an excellent indicator of the existence of an NFT in a given input as well as a good correcting rotation that should be made before starting further analysis.

7.2.5 Extended Corner Projection Analysis

Both row and column corner projections tend to produce distinct patterns which could be used to produce better projection "edges". After collecting a large amount of typical projections, analysis can be performed to find generalizations and ultimately a fast test which can be built into this method which not only provides better detection rates but also improved boundary selection.

7.2.6 De-warping of Skewed Inputs

Instead of fitting an aligned rectangle around the bounding Hough lines, better results may be produced if the four corners are "stretched" into an appropriately sized rectangle. This de-warping of the skewed input would produce results do not contain as much non-NFT area and are better suited for further analysis.

7.2.7 Segmented Hough Transformation

An alternative to Subsection 7.2.6 involves a much more intensive Segmented Hough Transformation which would divide the image into a grid of smaller segments and perform separate Hough Transforms within each segment [16]. The advantage to this technique is the ability to look for connected Hough lines between segments that could actually be skewed, curved, or even zig-zag lines. These non-straight lines cannot be detected by a regular Hough Transform but the performance penalty to do this would be quite high. The payoff, however, could be significantly improved ability to detect and de-warp oddly shaped NFTs.

7.2.8 Parallelization and Pipelining of Computations

As mobile devices continue to received increased memory and multi-core CPUs this optimization will become more relevant. Some steps in this method can be performed at the same time, so long as enough memory and bandwidth are available on the mobile device to actually see an improvement in speed. Additionally, it is possible to create multiple workers which could be staggered to analyse different images from different times and create the illusion of a faster overall frame-rate.

REFERENCES

[1] Altman, D. G., and Bland, J. M. Diagnostic tests. 1: Sensitivity and specificity. *BMJ: British Medical Journal 308*, 6943 (1994), 1552.

[2] American Academy of Ophthalmology. Eye health statistics. http://www.aao.org/newsroom/upload/Eye-Health-Statistics-April-20111.pdf, 2011. Retrieved March 14, 2013.

[3] Ballard, D. H. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition 13*, 2 (1981), 111–122.

[4] Canny, J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6 (1986), 679–698.

[5] Graham, D. J., Orquin, J. L., and Visschers, V. H. Eye tracking and nutrition label use: A review of the literature and recommendations for label enhancement. *Food Policy 37*, 4 (2012), 378–382.

[6] Helal, A., Moore, S. E., and Ramachandran, B. Drishti: An integrated navigation system for visually impaired and disabled. In *Proceedings of the Fifth International Symposium on Wearable Computers* (2001), IEEE, pp. 149–156.

[7] Kim, D.-S., Lee, W.-H., and Kweon, I.-S. Automatic edge detection using 3×3 ideal binary pixel patterns and fuzzy-based edge thresholding. *Pattern Recognition Letters 25*, 1 (2004), 101–106.

[8] Krishna, S., Balasubramanian, V., Krishnan, N. C., Juillard, C., Hedgpeth, T., and Panchanathan, S. A wearable wireless rfid system for accessible shopping environments. In *Proceedings of the ICST 3rd International Conference on Body Area Networks*

(2008), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), p. 29.

[9] Kulyukin, V., Gharpure, C., and Nicholson, J. Robocart: Toward robot-assisted navigation of grocery stores by the visually impaired. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2005), IEEE, pp. 2845–2850.

[10] Kulyukin, V., and Kutiyanawala, A. From shoptalk to shopmobile: vision-based barcode scanning with mobile phones for independent blind grocery shopping. In *Proceedings of the 2010 Rehabilitation Engineering and Assistive Technology Society of North America Conference (RESNA), Las Vegas, NV* (2010).

[11] Kulyukin, V. A., and Kutiyanawala, A. Demo: Shopmobile ii: Eyes-free supermarket grocery shopping for visually impaired mobile phone users. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2010), IEEE, pp. 31–32.

[12] Kutiyanawala, A., Kulyukin, V., Nicholson, J. Toward real time eyes-free barcode scanning on smartphones in video mode. In *Proceedings of the 2011 Rehabilitation Engineering and Assistive Technology Society of North America Conference (RESNA), Toronto, Canada* (2011).

[13] Laganière, R. *OpenCV 2 computer vision application programming cookbook.* Packt Publishing Ltd, 2011.

[14] Lanigan, P., Paulos, A., Williams, A., Rossi, D., and Narasimhan, P. Trinetra: Assistive technologies for grocery shopping for the blind. In *10th IEEE International Symposium on Wearable Computers* (2006), pp. 147–148.

[15] Li, H., Zheng, H., and Wang, Y. Segment hough transform–a novel hough-based algorithm for curve detection. In *Fourth International Conference on Image and Graphics (ICIG)* (2007), IEEE, pp. 471–477.

[16] Merler, M., Galleguillos, C., and Belongie, S. Recognizing groceries in situ using in vitro training data. *SLAM, Minneapolis, MN* (2007).

[17] Nicholson, J., Kulyukin, V., and Coster, D. Shoptalk: toward independent shopping by people with visual impairments. In *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility* (New York, NY, USA, 2008), ACM, pp. 241–242.