Abstract

While current systems already provide reliable results in correctly reading barcodes at close range, we want to increase the limits of these systems so they will be able to read these codes with extreme levels of blur and noise. Building off of the work by Christine Lew and Dheyani Malde[2], we propose a method to analyze a subsection of a barcode with an unknown signal and blurring function in order to learn their values. This function uses the Wiener filter to analyze and filter out noise and blur from a blurry barcode, while using a dictionary brute force method to try every possible combination of the first two digits of a barcode to find a blurring function and clean signal that created the original input. Through testing, our method successfully analyzes a subsection of a blurry barcode and returns a fragment of a clean barcode along with a kernel estimate similar to the actual function that blurred the image, possibly allowing future processes to accept even more limited data and correctly reconstruct the original data.

1 Introduction

Our goal is to improve current methods of barcode deblurring by learning the blur of a distorted barcode and discovering the clean signal.

1.1 Barcode Structure

A barcode has a specific structure, depending on what system it follows. For our research, we analyzed barcodes following the Universal Product Code structure, or UPC for short. This structure encrypts 12 numbers into 95 binary values of either black or white. Each of the 12 numbers is represented by some combination of 7 binary digits, and then the first six values are separated from the second six values of a five digit black-white-black-white-black combination. Finally, each end of the barcode is capped with a 3 digit black-white-black combination, for a total of 95 values.
1.2 Previous Methods

Common methods in the past used the total variation such as in [1] or regularization parameters to control the influence of noise when deconvoluting, or clarifying, blurry barcodes. Another method in [3] focused on using an algorithm that included the symbology of a barcode to decrease the complexity of the problem.

1.3 Our Method

We will use a combination of the dictionary approach and Wiener filter to decode barcodes with various levels of blurring and noise. Our method will consider the structure of a barcode as well as limit the amount of data we analyze in order to decrease the number of unknowns we must solve for. We analyze the first two numbers of a barcode by guessing its possible value and then finding a kernel to blur with it. With the reduction of the number of unknowns, we are able to shorten the analysis to a very brief moment. In exchange for these advantages, a resource intensive greedy algorithm must analyze every possible combination that the subsection can take. Despite these costs, we were able to reduce the deconvolution problem from $2^{95}$ values down to $10^2$, thus yielding promising results.

1.4 Modeling Method

In order to properly analyze a barcode, we will follow the previous methods of modeling a blurry barcode with this equation:

$$f = k * u + n$$  \hspace{1cm} (1)

The blurry barcode is represented by $f$. We can decompose this signal into three parts: $u$ representing a clean barcode, $k$ representing a blurring kernel, or function, that when convolved with $u$, creates the blurred barcode $f$. $n$ represents random noise induced through the process of taking the image of the barcode, such as artifacts through medium storage or image quality. This noise will either darken or lighten various parts of the barcode in a random manner.

1.5 Explanation

Because convolutions have the same effect on each element of a signal, we do not have to analyze the whole signal in order to find a proper blurring function. Instead, we can analyze a small portion of the signal at a time, and still find the function that was used to convolute the entire data set.

Another important factor is that there are a finite number of forms the signal may take due to the definitive nature of a barcode: each value of a barcode is represented by a certain pattern depending on its location in the barcode, and the barcode also has specific structures at various points that are consistent no matter what its values are. Because of these constraints, we greatly reduce the number of unknown barcode value we need to estimate from $12^{10}$ down to 100.
2 Barcode Generation

To test the accuracy of our systems, we synthesize our own barcodes to compare our results with the actual data used to create the blurry barcode. In order to achieve this, we use the same model in Equation 1 by generating each variable used: a base $u$ convoluted with and function $k$ followed by additive noise $n$.

2.1 Generating the Base

We encode a 12 digit value using the UPC table of values to create $u$ as shown in Figure 1a. After creating the barcode, we stretch it to simulate the signal of a large image of a barcode, Figure 1b. The amount we stretch by is determined by a stretching factor, a variable we set at the beginning of the program. Lower values stretch the barcode less than higher values.

2.2 Generating $k$

Afterwards, we generate a Gaussian function to use as our synthetic $k$, because this would simplify $k$ to have only one unknown to solve for and convolute the result with our synthetic $k$. Figure 2a represents our generated $k$, while Figure 2b represents our $u$ convoluted with $k$.

2.3 Adding Noise

Finally, we add noise to each of the 95 values by adding it with a randomly generated value based on a normal distribution. This distribution is centered on 0 and has a radius defined by $n$. Figure 3a charts the additive noise at each point on the convolution, while Figure 3b charts the result.

Now we have a synthetic $k$, $u$ and $f$ that we can compare our results to at every step of our process, with controlled values for the Gaussian function $k$, the generated $u$, and the variance of the noise distribution $n$. 

Figure 1: Notice the x limits on each figure
Figure 2: Stretched barcode from Figure 1b convoluted with k from Figure 2a

Figure 3: Noise is added to every point of the convolution
3 Model Verification

Another key part of our method is to determine how well our synthetic data represented real images of barcodes. With this, we can determine how accurate convoluting barcode data with a Gaussian blur kernel represents real world blur, as well as determining the necessary standard deviation for the Gaussian blur that would achieve real world results.

3.1 Image Capture

We gathered information by taking pictures of real world barcodes from various products with a Digital Single-Lens Reflex camera, and blurred the image by manually adjusting the lens to acquire an unfocused image, some examples in Figure 4.

3.2 Image to Graph

We send this blurred image through MATLAB to acquire a binary graph where 255 represents black, a zero represents white, and a value in between represents a shade of gray. The values throughout this graph range from 0 to 255 because of the way grayscale is encoded in MATLAB, so we divide the values by 255 to achieve a scale from 0 to 1, the way we represent our synthetic barcodes in our simulations.

3.3 Extract Stretching Factor

Now that the scale is correct, we must shrink the image to match the width of our synthetic data. We know that a barcode can be minimally represented by 95 digits, so we must shrink our graph to fit into this size. In order to properly shrink the barcode, we must know where the barcode starts and ends in the graph. To accomplish this, we detect the first and last local minima of the graph, as shown in Figure 5. These local minima correspond to the middle of the first and last barcode digit respectively, so we know that there are 94 digits between these two locations: the middle of a bar is half a bars width from the edge, so with two ends, we are missing a whole bars worth of data from a
3.4 Extracting Barcode

Afterwards, we cut out the barcode starting from the first local minima to the last, and append zeros to each end to achieve the image in Figure 6. The number of zeros we add is based on the stretching factor, enough so that the algorithm would realize that the zeros are the ends of the barcode and not another digit. We use the Kronecker Tensor Product of an identity matrix the size of 95 plus the number of zeros we want to add at each end, and a vector of ones the length of our stretching factor to obtain a matrix the proper size to hold the cut barcode with enough zeros at both ends to ensure there is no confusion in our algorithm. The cut barcode is placed in the middle of this Product, and then the Product is inverted by subtracting each value from one. This is so we are consistent with how the bars are represented in our simulations where a zero represents black, a one represents white, and a decimal represents a shade of gray somewhere in between.

3.5 Histogram Analysis

Finally, we increase the contrast between the black and white barcode values in order to achieve a cleaner signal so that the values would be easier to read. To accomplish this, we create a histogram of our data and then push the darker values to zero and lighter values to one. By solving a system of linear transformations, the white and black areas of the barcode become easier to distinguish. Figure 7 shows the before and after results of this analysis.
Figure 6: Extracted Barcode before placed into Product

Figure 7: Notice where the peaks fall along the Y-axis
Finally we compare this signal from an actual image with a barcode we generated synthetically. We generate our barcode to encode the same 12 numbers the real world barcode encoded, and then find a $k$ that when convoluted with our barcode would result in a $f$ that is similar to our result from the real world barcode. We compare our accuracy by determining an error defined by the Euclidean distance between our $f$ and the results of our real world barcode. With a small blur in the real world barcode, the error of our synthetic $f$ is 3.8335; repeating this process with the same barcode but with a large blur, our error results in 7.1212. Considering that this error is the buildup of small differences across a several thousand pixel wide image, our synthetic $f$ can be used as an accurate model of real world barcodes. A comparison of the before and after processing appears in Figure 8.

3.6 Comparison with Raw Data

Figure 8: A comparison of our interpretation. Red is our processed data, while blue is our raw data.
4 Method

Our method for generating our estimations for what \( k \) and \( u \) can be in order to achieve \( f \) follows the greedy algorithm by trying every single combination possible and calculating how close that estimation is to the actual results. In order to accomplish this, we use the following equation based on Equation 1:

\[
\min_u ||Ku - f||^2 + \lambda ||u||^2
\]  

(2)

The variables are similar: \( k \) represents the matrix form of the blurring kernel \( k \), \( u \) represents the vector form of the original signal. \( \lambda \) represents our regularization parameter and is used to control the influence of noise on our equation. The difference between our convolution with the convolution of our estimated \( k \) and \( u \) with the actual blurry data \( f \) will serve as an indicator as how close our estimates produced a result similar to the actual data. We return the \( k \) and \( u \) that gives us the closest value to \( f \) as our estimates for the actual \( k \) and \( u \) that when convoluted together gives \( f \).

4.1 Generate a Base

We start by generating the combination 00 and using this as our \( u \), calculate a \( k \) that when convoluted with 00 would result in \( f \), as shown in Figure 9.

4.2 Find \( k \)

To estimate the \( k \) needed to convolute with each value we test, we analyze only the first two numbers of the barcode, a total of 17 digits. We take the first two numbers of the full blurry barcode and treat that as our \( f \). Because now we know \( u \), \( f \) and \( \lambda \) in equation 2, we generate an arbitrary base \( k \) to start off our process and convolute this with our \( u \). It is expected that the difference between this convolution and our actual \( f \) will be large, so we refine \( k \) using gradient decent. After the refinement no longer produces significant changes to \( k \), we return the result as our estimated \( k \).

4.3 Calculating Error

We then calculate the error by taking the Euclidean distance of our estimated \( k \) convoluted with the combination 00 to \( f \):

\[
\text{error} = |Ku - f|
\]  

(3)

After this step we repeat this process with 01, Figure 10. If the error of convoluted 01 is less than the convoluted 00, we remember 01, its estimated \( k \) and the calculated error; and if not, we retain 00 and the associated \( k \) and its error.
Figure 9: The value and necessary $k$ to achieve input $f$ if $u$ was 00

Figure 10: Same as Figure 9, but with 01
4.4 Repeat
We continue on from 02 to 99, remembering the closest result to our blurred barcode. Once this process is completed, we return the $k$ value that when convoluted together results in the lowest Euclidean distance with our $f$.

4.5 Deconvolute
Now that we have $k$, we treat this problem as a non-blind deconvolution problem and use the Wiener filter to deconvolute the full $f$ with $k$ to find $u$.

4.6 Regularization Parameter Estimation
The regularization parameter is used to control the influence of noise in our calculations. We represent this parameter as $\lambda$, and it is currently set its value at the start of our analysis. The best value would be one that would set the second half of Equation 2 to zero.

5 Results
After repeated iterations, our system works well for at least moderately stretched barcodes with Gaussian blurs, with up to moderate levels of noise. By varying the values for the stretching factor and noise variance, we find widely varying performance and accuracy of the program.

5.1 Low Noise Level
With a stretching factor of 1, and an $n$ of 0.001, the system performs poorly. Although it does estimate a $k$ and a $u$ that does fit $f$ when convoluted, it does not estimate the correct values as shown in Figure 11, and takes an extensive amount of time.

With a stretching factor of 3, and the same $n$, the system performs much better, returning the correct $u$ and a $k$ that is close to the actual value, as well as completing much faster, as show in in Figure 12.

A large stretching factor of 5 continues to increase the accuracy of the estimated $k$ as well as returning the proper $u$ in Figure 13.

5.2 Medium Noise Level
With a stretching factor of 1, and an $n$ of 0.01, the system performs similarly to when the stretching factor was 1 and $n$ was 0.001. Although it does estimate a $k$ and a $u$ that does fit $f$ when convoluted, it does not estimate the correct values as shown in Figure 14, and takes an extensive amount of time.

With a stretching factor of 3, and the same $n$, the system performs much better, returning the correct $u$ and a $k$ that is close to the actual value, as well as completing much faster, as show in in Figure 15.
Figure 11: A low stretching factor and little noise does not return the correct $k$, but does return a similar $f$.

Figure 12: A moderate stretching factor and little noise returns the correct $k$ and similar $f$.

Figure 13: A large stretching factor and little noise returns the correct $k$ and similar $f$. 
Figure 14: A low stretching factor and some noise does not return the correct \( k \), but does return a similar \( f \).

Figure 15: A moderate stretching factor and some noise returns the correct \( k \) and similar \( f \).
Figure 16: A large stretching factor and some noise returns the correct $k$ and similar $f$.

Figure 17: A low stretching factor and high noise does not return the correct $k$, but does return a similar $f$.

A large stretching factor of 5 continues to increase the accuracy of the estimated $k$ as well as returning the proper $u$ in Figure 16.

5.3 High Noise Level

With a stretching factor of 1, and an $n$ of 0.1, the system performs similarly to when the stretching factor was 1 and $n$ was 0.001. Although it does estimate a $k$ and a $u$ that does fit $f$ when convoluted, it does not estimate the correct values as shown in Figure 17, and takes an extensive amount of time.

With a stretching factor of 3, and the same $n$, the system performs much better, but show significant defects compared to the simulations with low or some noise, returning the correct $u$ and a $k$ that is close to the actual value, as well as completing much faster, as show in in Figure 18.

A large stretching factor of 5 has about the same accuracy as a lower stretching factor, returning a poor estimated $k$, yet still returning the proper $u$ in Figure
Figure 18: A moderate stretching factor and high noise returns the correct \( k \) and similar \( f \).

Figure 19: A large stretching factor and high noise returns the correct \( k \) and similar \( f \).

5.4 Explanation

Our algorithm forces the result of \( k \) convoluted with \( u \) to be close to \( f \), and because of this, a lower stretching factor forces our \( k \) to take on more extreme values to fit within the narrow window of \( f \), with many repetitions. This would explain why the low stretching factor simulations require so much more time, because the process requires much more refining in order to find a \( k \) that can be convoluted with a \( u \). Because of these extreme values, wild values of \( u \) are used as our best estimate, even though it would clearly be wrong.

As the stretching factor grows larger, this allows more room and flexibility for our program to find a proper \( k \) to fit the proper \( u \) in order to recreate \( f \). With more data to analyze, more equations are made available, and with more values to work with, the system can quickly home in on a properly refined \( k \) in
a shorter amount of time.

However, as the level of noise increases, the more \( k \) must change in order to compensate the variation caused by the random noise \( n \). This is the cause of the increased deviation as our stretching factor grows larger as well as more significant noise levels. Despite these changes, our algorithm does return a close enough \( k \) to guess the correct first two digits of the barcode, showing the robustness of our method against noise.

6 Conclusion

Our algorithm has shown that it is possible to discover a blurring kernel when only a blurry barcode is provided. Not only is this possible, but we can also discover at least the first two digits of the barcode. This method works against high noise level as well.

Our future goal is to improve the accuracy of our algorithm. One method is to improve the \( \lambda \) estimation: instead of having it preset at the beginning of our analysis, we would base it on the level of noise in the barcode. To accomplish this, we would analyze a portion of the barcode where consecutive blocks of black or white to determine the variance of noise, and base \( \lambda \) on that value.

References

