Mobile Augmented Reality for Grocery Shopping

Monica Thompson
Department of Computer Science
University of Wisconsin – Eau Claire
Eau Claire, WI 54702
thompmon@uwec.edu

Daniel Stevenson
Department of Computer Science
University of Wisconsin – Eau Claire
Eau Claire, WI 54702
stevende@uwec.edu

Abstract

This paper presents the development of an augmented reality application for a mobile phone. Techniques from computer vision, image analysis, and mobile computing are used to provide the augmentation. Specifically, the focus is centered on the development of a program for the Android OS 2.2 on HTC's Evo 4G that will capture and analyze image information and create a real-time augmented reality overlay. The program utilizes the Evo's camera for image capture and the Android SDK and OpenCV for image analysis and generating the augmented reality.

1 Introduction

Augmented Reality is the process of superimposing computer generated information on top of actual pictures or video in real-time. With the increased use of smart-phones that contain quality cameras capable of taking video there has been a recent increase in the development of these types of applications. Normally the capturing of picture/video information is fairly straightforward as is the overlay of information onto the screen. The complex issues are in the identification and tracking of objects in the images. These are classic problems in computer vision that have found a new home in augmented reality for mobile phones.

In this paper, we present a specific mobile augmented reality application. The specifics of this application are discussed in section 2. Section 3 discusses the details of our implementation. The hardware and software layers we used are presented in Section 4. We conclude with some information on future work we are currently engaged in.

2 The Cereal Box Identification Application

The specific application we developed is for augmenting cereal box information in a grocery store setting. When activated, the application prompts the user to input a brand of cereal that they wish to search for. The camera must then be focused on a set of grocery store shelves. Once this is done the application proceeds to find boxes of the specified brand. This kind of functionality's usefulness is not limited to just searching for a specific brand of cereal. It can also be applied to general cases such as searching for cereal with good nutritional content or avoiding cereals containing specific allergens. In conjunction, additional nutritional information can be presented alongside the cereal.

The application starts out by acquiring the current frame of the camera and locates the grocery store shelves in the image. This does two things. The first is that it helps gauge the size of the cereal boxes from where the user is currently standing. Given an average height to width ration of a cereal box, the application determines how big cereal boxes will appear in the picture, assuming the height of a cereal box takes up a fixed portion of the space provided on the shelves. This is a fairly good assumption to make given how grocery stores normally stock the shelves. The second function this serves is dividing the frame into organized regions. This allows the application of search only certain regions in chunks, rather than traversing the image pixel by pixel.

After this step has been completed the application then begins searching for specific boxes using both pixel and histogram information of the brand the user indicated. This is a complex process that can lead to false positives so our implementation had to pay careful attention to this step.

Once the application has finished searching, it goes through the results and draws them on the frame. The frame is then returned and displayed on the phone's camera, giving the

appearance that the boxes have been outlined in reality. Additional information about the cereal in question can also be displayed on the screen at this point if desired.

3 OpenCV Implementation

OpenCV [1,7] library provides a number of different functions that utilize different image manipulation techniques. This is useful because much of the application functionality is complex and OpenCV helps provide a good basis for the operations needed. Such operations include camera calibration and image capture, isolating the shelves using horizontal lines, and for finding the correct cereal box matches using template matching and color distribution.

3.1 Acquiring Frames

The initial camera capture is done using the function cvCaptureFromCAM. A frame is then pulled from the camera using cvQueryFrame by passing it the captured camera. Once this is done the frame can then be manipulated like any another image that would be typically loaded from file [1]. Use of these basic camera capture functions is fairly straightforward.

3.2 Isolating Shelves

The next portion of the application concentrates on finding the shelves in the image. This is important so one can then determine where to look for cereal boxes. We have approached this problem by using edge detection. There are many options but the cleanest method appropriate for this project was to start with a Canny edge detector. Canny is a classic approach to finding all the edges in a given image.

When cvCanny is run it takes in two images (the original and an image to write to) as well as thresholds to determine the amount of detail render. Canny writes to a blank image which, when displayed, appears as a black and white image. The edges are highlighted in white and fill appears as black [1]. The Canny edge detector analyzes each pixel gradient and determines whether or not it is an edge pixel. If it falls within the range of two specified threshold arguments that are passed into the function then it is marked as an edge pixel by recording a non-zero pixel value. This highlights the pixel in white in the resulting image. Non edge pixels are recorded as zeros, thus showing up as black. Essentially what is returned is a binary visualization of the image [3]. The results of a typical scene can be seen in Figure 1. It is important to set the thresholds well for the given scene or else one may end up with either too little detail or too many needless edges. Luckily, most grocery stores use similar florescent lighting conditions so we have not had much trouble with having to tune these values for different lighting conditions.

Figure 1: OpenCV Canny Edge Detector.

The result of OpenCV's Canny edge detector is simply an image where the edges have been highlighted. Since we need actual lines and points from the image in order to draw our results, the next step is to narrow down which highlighted edges are important. This is done by implementing a variation on Hough Lines (cvHoughLines2). The Canny edge detector is essential for running Hough Lines in OpenCV since it draws lines based on whether or not a pixel contains a nonzero value [1]. The result of a standard Hough transformation can be seen in Figure 2. Notice that there are far more lines than one might imagine for a typical image.

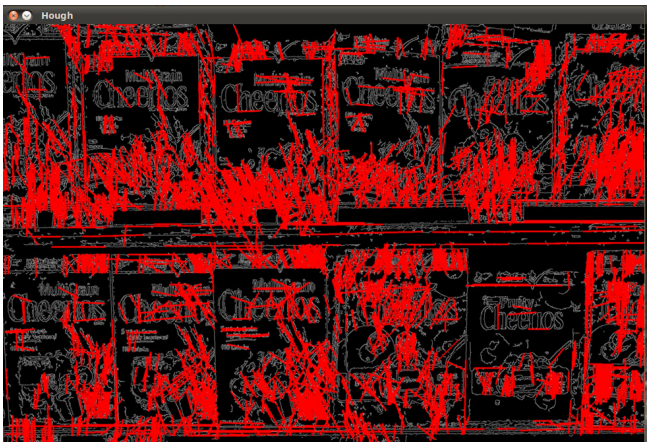

Figure 2: OpenCV Hough Lines.

The next step is to determine which line information is meaningful and which line information is simply to be considered noise. We have been applying multiple thresholds to cull lines that we have determined cannot be shelf lines. In particular, removal of short lines reduces the noise by a significant amount. We assume shelf lines are also oriented horizontally so we look at the orientation of the lines as well. By grouping long horizontal lines together we can get a good indication of where the shelves are located.

Once shelf lines are determined, the distance between them is calculated. This returns a height which is used to generate a width. This is done by multiplying it with a pre-calculated ratio of a cereal box, giving us a good idea of what the average cereal box dimensions are in the image. With this, the application is able to start searching the shelves for specific boxes.

3.3 Box Searching

We employ a two stage box searching algorithm. The first stage uses template matching to try and find pixel matches in the acquired image. This gives us candidates for potential boxes. However, false positives are an issue to consider since the box being searched for is sometimes not in the acquired image. Thus, we use a second histogram stage to help refine the results.

3.3.1 Template Matching

In this stage we use OpenCV's Template Matching algorithm (cvMatchTemplate) to try and find a template image in an acquired image. The algorithm does this by effectively sliding the template over the input image and then, using a matching function, it determines how good the comparison is at each location in the image. You can see an example template and acquired image in Figure 3.

There are several matching functions that can be used. After trying several of them, it was decided that correlation coefficient (ccoeff) matching produced the best results. The ccoeff method matches a template relative to its mean against a section of the acquired image relative to its mean. Thus, a perfect match will produce a 1 and a perfect mismatch will produce a -1. A 0 means there is no correlation [1]. The results of this matching are shown in Figure 4. Note that two bright spots can be seen where the two shredded wheat boxes show up in the acquired image.

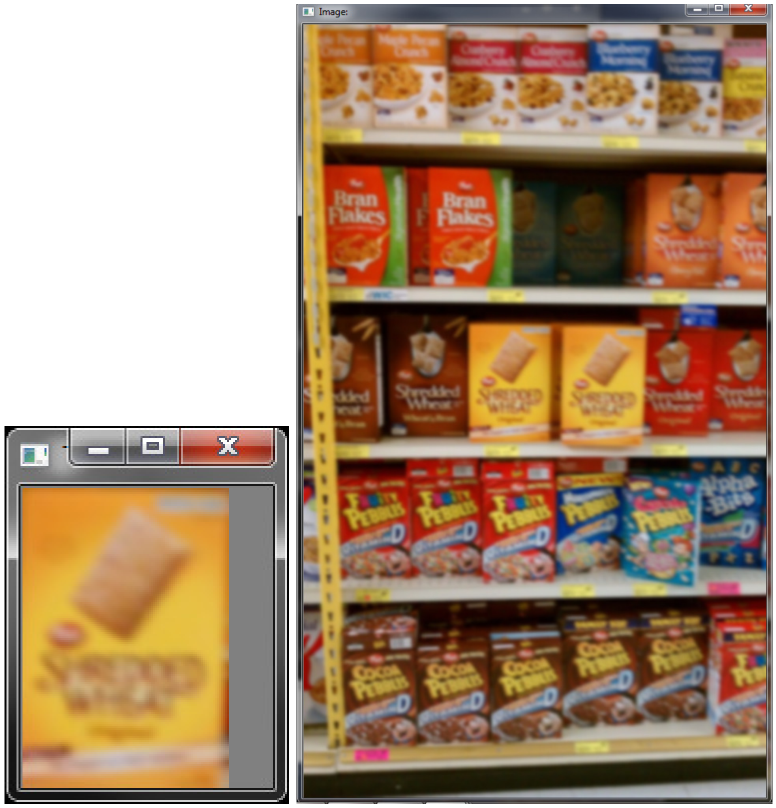

Figure 3: A cereal box template (left) and an acquired image (right)

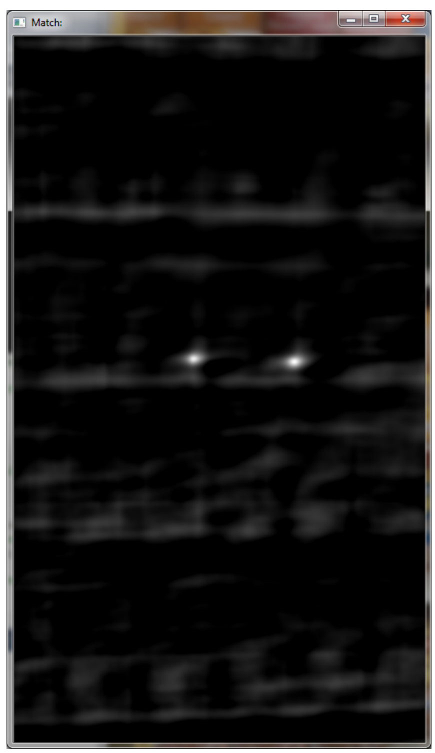

Figure 4: Results of ccoeff matching

Note two things with this matching approach. First is that actual pixels are being matched. This implies that if the pixels in the template are not exactly the same as in the acquired image it will cause lower matching scores to result. To help minimize this and to clean up noise, both the template and acquired image were blurred slightly (hence the apparent fuzziness in Figure 3, Figure 5, and Figure 6). The second implication is that the template image needs to be the same size as the cereal boxes in the acquired image for the matching to work. We simply resize the template image to be close to the correct size based off of the shelf height information we obtained earlier.

Once the matching values have been determined, they are searched to find the highest match. This, in theory, is where the match occurred in the image and is marked and passed onto the final display stage where it is marked and augmented with extra information. For the example above, the results of finding the highest match are shown in Figure 5.

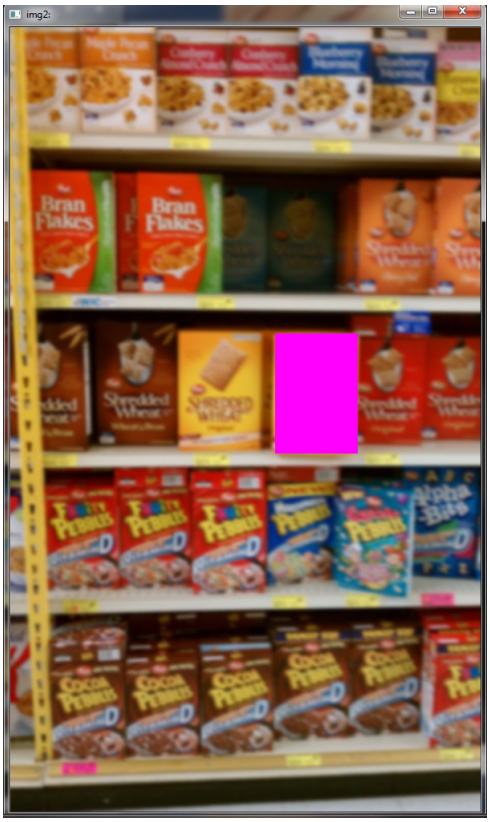

Figure 5: The pink area is the matched region of in the acquired image

However, the highest match does not always mean a proper match. In particular, if one is searching for a template that does not exist in the acquired image then no match should be found. A couple of things are done to determine if the match found is a correct match

or not. The first is to examine the location of the proposed match. Often false matches occur by combining parts of boxes from two shelves. Since we know where the shelves are approximately located we can quickly determine if the match could cross shelf boundaries and thus whether or not it should be ruled out. See Figure 6 for an example.

Second, we can also look at the matching values. Good matches tend to be in the 0.75 – 1.0 range. However, that assumes the same lighting conditions for the template and acquired image as well as template size. If the lighting conditions are a bit different these match values can drop to 0.5 – 0.75. This is not a terrible situation since most false positives tend to be in the 0.0 – 0.5 range. Many false positives also cross shelf boundaries.

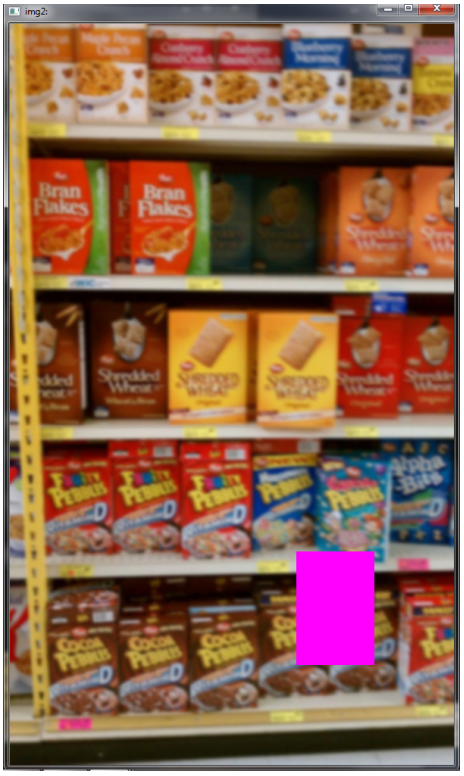

Figure 6: The best match crosses shelf boundaries when looking for a box that is not present in the acquired image.

3.3.2 Histograms Matching

There is one more factor that plays an important role with cereal boxes in particular. Companies often change the boxes slightly from month to month to advertise new promotions. Thus, the basic box looks similar but is different in subtle ways. For instance, the characters on the boxes have changed positions or an extra set of words has been placed over the top. These subtle changes are hardly noticed by humans, but they do have an impact on the template matching since the underlying pixels are certainly not

the same in portions of the image. We still can get a decent template match, depending on how much has changed, but matches in the range of 0.25 – 0.5 are not uncommon. Thus, we use a second histogram test which checks the underlying colors in the potential match area to see if they are similar to the colors in the template.

OpenCV has support for histogram matching. In fact it has a fairly robust set of histogram creation and matching methods. We use the relatively simple cvCompareHist to compare the two computed histograms using an intersection method. Just like with template matching this gives a number close to 1.0 if the histograms match well and a lower number if they do not.

Since histograms divide the colors into bins over a particular region, as long as the general color distribution of the boxes are the same then a good match will be obtained. This should be true even with some subtle modification to the design of the box. The company does not want the look of the box to change much to the human eye after all.

When constructing histograms, we found that using Hue, Saturation, Value color space worked better than Red, Green, Blue color space. That is because Hue captures the overall color of a pixel and the Saturation and Value channels focus more on the lighting conditions and darkness of the ink. Thus, if one just constructs the histogram out of the Hue channel it makes the histogram matching more immune to changes in lighting [2].

When matching exact boxes under exact lighting conditions we got histogram matches in the 0.75 – 1.0 range which validates the information obtained in the template match stage. If the box has changed slightly or the lighting is off a bit then we still often get histogram matches in the 0.5 - .75 range, which can also be used as positive feedback to confirm that the template match is correct. It should be noted that we do get some false positions still, as many cereal boxes do have similar sets of bright colors and some box variations within a brand sometimes vary more than subtly.

3.4 Displaying Results

After the frame has been fully analyzed and results have been gathered, the program will then draw results onto the frame. Each match found (within a certain range of error) is stored in memory for later use. The program will access these results and draw them on the frame. The modified frame is then returned to the camera and then the camera displays it. This gives the appearance that the matching cereal boxes have been highlighted in real time.

Figure 7: Displaying matches.

4 Compiling Application to Run on Android

The software aspect of the project makes use of the OpenCV 2.0 and utilizes tools such as Eclipse 3.5 Galileo, Android SDK revision 8, Dmitry Moskalchuk's modified version Android NDK revision 4, and SWIG.

Since OpenCV is a C++ library and the Android SDK only supports Java, this project also uses SWIG and Dmitry Moskalchuk's modified version Android NDK. SWIG is a tool that's used to connect C++ libraries with other programming languages which, for our purposes, also includes Java [4]. Also aiding in this is the Android NDK, an extension of the Android SDK that allows a user to develop in C/C++ [5]. Moskalchuk's modifications to the NDK allow for additional support for C++ exceptions, RTTI, and the standard C++ library [6].

Figure 8: HTC Evo 4G.

For hardware this project uses HTC's Evo 4G. The decision to use the Evo 4G over other models was based on the fact that it supported the most current version of Android at the time, Android 2.2. At this point in time there is no upgrade available update to Android 2.3. The phone is also equipped with an 8.0 megapixel camera, which is necessary for high quality image capture.

The process of copying the application onto the phone is fairly simple, with the proper installation of the Android SDK. For Windows, it's a simple process of connecting the phone to the machine and then using the Eclipse Android SDK plug-in to run the application from the phone instead of the emulator. Windows plug and play should already have the phone identified. For Linux (specifically Ubuntu), the android debugger server must be started before Eclipse is opened. Once the server is started it must be commanded to listen to the USB ports. This detects the model of the phone (which is essential for installing the application onto the phone). After that you simply run the application from the phone the same way you would in the Windows environment. Installing the application from a command prompt is also an option as well.

5 Future Projects

This project is large and diverse enough that it provides enough open ended questions and possibilities for further research. One potential project would involve modifying the current program to track the cereal boxes it identifies, rather than reanalyzing each additional frame and relocating each box. This would speed up the computer vision analysis as well as make it less prone to false positives.

Another future project would involve handling images taken at an angle. Currently, the functionality of the application only works in a setting in which the user is directly facing the grocery store shelves they are interested in. In this setting the shelf heights and lengths are fairly consistent, however if taken at an angle the height of a shelf appears to vary in a 2D picture. This makes it hard to gauge exactly how big the box needs to be when highlighting and drawing our results onto the frame, as well as comparing histograms and templates.

There are also many additional computer vision algorithms available that could be brought to bear on this project. A more in depth analysis of the matching results could be performed and then compared to matching results from other matching approaches.

6 Summary

At this point in the project we have accomplished basic template matching and analysis of color distribution using histograms. The project also has functioning line detection and region isolation. There are many complex pieces in this project; however, not all of them function well together as a whole. Each piece works sufficiently well on its own but combined the entire process takes a long time to locate the box it is looking for. Many steps are required in order for the identification process to run smoothly but due to the number of steps and the large amounts of calculation performed in each step this project has not been able to produce a result in real time.

References

[1] Gary Bradski & Adrian Kaehler. Learning OpenCV. *O'Reily Media, Inc.* September 2008.
[2] Robert Haralick & Linda Shapiro. Computer and Robot Vision vol I and II. *Addison Wesley*. 1992.
[3] John Russ. The Image Processing Handbook. *CRC Press*. 1995.
[4] http://www.swig.org/
[5] http://developer.android.com/sdk/ndk/overview.html
[6] http://smartctl.net/android/ndk.php
[7] http://sourceforge.net/projects/opencvlibrary/