Automated Coin Detection on Android Phone

Mihir Pendse
Department of Electrical Engineering
Stanford University

Yiwei Wang
Department of Electrical Engineering
Stanford University

Abstract—The proposed algorithm for automated coin detection uses a Hough transform to determine the radius of coins while exploiting features such as the color of the penny. Overlapping coins and the presence of clutter are tolerated though the angle of the camera and the color of the background are restricted. Though the algorithm as it stands is slower than a human performing the same task, it is conceivable that in future work the methods described here could be scaled to a point where they find a real-world application in eliminating the need to manually count coins.

I. INTRODUCTION

Automated coin detection using a mobile platform such as an Android phone could have practical value in eliminating the tedious task of determining the monetary value of loose coins by hand. Various techniques have been developed to implement coin recognitions based on feature matching [1,2,3,6]. Feature matching for coin recognition has been shown to work well with eigenspace [1], edge [2], and gradients [3]. Although all these approaches achieved more than 90% overall accuracy in coin recognition [4], the drawback is that the large amount of reference coins required makes such techniques impractical for mobile applications, especially considering the variety of patterns that appear on US coins. Moreover, these matches require the references to be on the same scale as the coins. Text recognition based identification [5] and SIFT based matching [6], are scale-invariant but require a rather high quality image, which might not always be attainable in mobile applications.

Here we describe an algorithm that successfully determines the value of a collection of the four standard US coins: pennies, nickels, dimes, and quarters. Instead of a standard imaging matching approach using feature detection with SIFT or SURF, certain features of this particular identification task are exploited to develop an approach that is more robust and less sensitive to lighting variations and glare from coins that would make a typical image matching approach infeasible. In particular, the limited number of coins to be detected (four), the distinction in color of the penny from the other coins, and the difference in the radii of the coins allows automatic determination of the scale of the image and subsequent detection of the coins. Only in special case boundary conditions such as the absence of pennies, does SIFT need to be used in order to establish the scale of the image. The proposed algorithm involving the Hough transform for radius determination can function with adjacent coins touching one another as well as with the presence of clutter that is not circular in shape. Limitations of the implementation are that a colored background significantly different from the color of the coins (such as green or blue) is needed, the camera must be reasonably parallel with the plane of the coins, and the running time of the algorithm is sacrificed for the sake of accuracy, though possible improvements in all these areas in future work are discussed.

II. PROCESSING FLOW

An image taken using the Android phone is sent to a server for processing using MATLAB and the computed dollar value of the coins is sent back to the phone to be displayed on the screen. An example of an unprocessed image taken with the phone is shown in Figure 1. The various processing steps performed by MATLAB are described here.

A. Thresholding

The color image taken by the phone must first be converted to a binary mask with the coins in the foreground. To do this, the hue and saturation properties of the coins compared to the background are exploited. As seen in Figure 1, both the hue and saturation values of an unprocessed image have sharp peaks at a hue value of about 0.35 and saturation value of close to 1. A convenient way of incorporating both hue and saturation in determining a threshold is to determine a threshold rating using the following formula:

\[ \text{threshold} = \max(h, s) - \text{pedest} - \text{scale} \times h[x, y] - \text{pedest} \]

This will give an image of the threshold ratings which can then
be converted to a binary mask by using a threshold that minimizes the mean square difference between the threshold and the threshold rating. Any irregularities in this thresholded image can be eliminated by a close operation followed by hole filling and removal of regions with areas less than some threshold. It should be noted that in this binary image overlapping coins and clutter with area greater than the area threshold may be present. An example of a thresholded image with clutter in the bottom left of the image is shown in Figure 3.

B. **Hough Transform**

To determine the radius of each circle in the thresholded image, an edge detection is performed on the thresholded image, and a Hough transform is applied to this edge image. The Hough transform is specific to a circle of given radius and the result of the transform gives a likelihood of each pixel being the center of a circle with the given radius. In order to determine the radius of each circle in the image, the Hough transform for a range of radii is computed and the radius corresponding to the maximum intensity at each pixel is determined. If this maximum intensity exceeds some threshold, the pixel is assigned the value of the associated radius and if the maximum intensity is below the threshold, the pixel is assigned a value of 0. A closing operation is then performed to produce a closed region of radius values centered at the center of each circle. Area thresholding is then performed to eliminate noise. The centers of the clutter in the thresholded image should be 0 in the radius image if the clutter is not circular. Additionally, overlapping circles forming a single region in the thresholded image should produce distinct regions in the radius image. An example radius image is shown in Figure 4.

From the radius image, a new thresholded image with no
clutter and a distinct label for each coin regardless of overlap can be produced by drawing a circle of the appropriate radius around the centroid of each region in the radius image. An example of this new thresholded image is shown in Figure 5.

![Thresholded Image without Clutter](image)

**Figure 5: Thresholded Image without Clutter (testing16)**

C. **Penny Identification**

The new thresholded image can be used to identify pennies on the basis of hue and saturation. As is suggested by the histogram of the hue and saturation values of the coins in Figure 6, a threshold that minimizes the mean square difference between the data points and the threshold can be determined to differentiate pennies (which usually have hue values below about 0.1 and saturation values above about 0.5) from other coins.

![Saturation Image (without Background)](image)

**Figure 6: Histogram of Hue and Saturation Values of Coins (testing16)**

Once pennies have been identified their radius can be computed to establish the scale of the image and determine the radii of all other coins in pixels based on their ratio to the radius of the penny. In the special case where there are no pennies in the image, an alternative method is needed to establish this scaling.

D. **SIFT matching**

If no pennies are found, SIFT matching is used to determine scaling. The method used does not involve finding a rigorous match between each coin and a member of a template database as is usually done with SIFT image matching. Instead, a quick approximation of the number of common feature points between each coin and each member of the database is computed in with high tolerance for false positives and false negatives. This tolerance is acceptable because it is in only necessary for there to be one strong match among all the coins as one match is sufficient for determining the scaling and thereby computing the radii in pixels of all the other coins. Despite the lack of rigor of the approach, it was found that even with a single coin, a correct match was obtained almost half the time. In most cases the SIFT algorithm may never actually run due to the presence of a penny.
E. Dime Identification

Once the radius of the penny has been determined either directly or through SIFT matching and subsequent scaling, the dimes can be identified as all coins with radii less than the radius of the penny. Finding dimes in this manner is very robust since dimes will always have a smaller radius than pennies even though their actual radius in pixels may not be exactly what is predicted through scaling of the penny radius. In this way two of the four types of coins have been identified without having to know the absolute radius of the coin in pixels.

![Figure 8: Image with Dimes Identified (testing16)](image8)

F. Nickel and Quarter Identification

Given the radius of the penny and dime coins, the radius of the nickel and quarter coins can be estimated. Making this estimation after dime identification provides a more robust estimate than doing so immediately after penny identification because both the penny radius and dime radius, which possess nonredundant information, can be used in the estimation. Once the estimated nickel and quarter radii are determined through scaling of the penny and dime radii, the identity of each remaining coin can be determined based on whether its radius is closer to that of the nickel or quarter.

![Figure 9: Image with Nickels Identified (testing16)](image9)

III. ANDROID IMPLEMENTATION

Although the original idea was to implement the entire program flow in OpenCV once the MATLAB script functioned properly, it was soon discovered that the runtime of the algorithm takes about 1 minute in worst cases. Considering the fact that Android mobile device possess considerably lower computing power compared to a PC, it seemed logical to let the PC handle the computation and implement a server-client setup for the application, where the mobile device captures the image of coins and uploads it to the server requesting to start the MATLAB script. The server receives the request, processes the image with the MATLAB script and returns the value of the coins back to the Android device for display. A flow chart showing the interaction between the Android device and the server is shown in Figure 11.

![Figure 11: Program Flow of Android Client and Server](image11)

The infrastructure of the client server interaction between the Android device and the server is based on the source code provided in the class tutorials. Modifications are made to suit the need for the implementation. As the MATLAB experiment shows, the ability to correctly identify coins is largely based on the quality of the image. Although in some cases, especially when pennies are present, the tolerance on image quality is rather high as we can still extract the radii of the coins proportionally regardless of the quality of the details. However, once we turn to SIFT matching in the absence of pennies, the quality of the image became crucial for identification of the coin.
to establish reference as described above. The autofocus function is added to the camera of the mobile device to ensure the features on the coins are captured for SIFT matching purposes. Another modification is to send the data of coin values directly from the server to the mobile device through stream to minimize the data transfer requirement for the application.

IV. EXPERIMENTAL RESULTS

A. Overall Performance

Though the runtime of the implemented algorithm is quite long from about 20 seconds to 1 minute, an accurate result is produced for many situations. It was found that up to 15 coins could be present with a correct calculation of their value as seen in testing16, the image used in the “II. Processing Flow” section. Here the number of coins is limited by the field of view and focus of the camera used rather than an inherent limitation of the algorithm. The algorithm was also seen to be insensitive to the touching of coins. Though these adjacent coins form a continuous region in the thresholded image they still produce distinct regions in the radius image because the Hough transform is able to detect their centers. The algorithm was also shown to be insensitive to clutter as long as the shape and color of the clutter was distinct enough from the shape and color of the coins. This is because thresholding successfully eliminates clutter with different saturation and hue properties than the coins and the Hough transform is able to eliminate non-circular shapes from the image. Additionally, the background need not be a fixed color. Any color is acceptable for the background as long as its spectrum does not overlap with those of the coins. In the test images, both a blue and green background were used with success. The experimental results can be reproduced by the reader by simply running the provided MATLAB script titled coin_counter, selecting one of several test images included in the zip file, and stepping through the figures produced for the various stages in the processing flow (it is assumed that the user has access to the vlfeat-0.9.14 library for SIFT functionality). Some examples of representative test images with the correctly determined dollar value in the title are shown here:

B. Performance on Mobile Device

The implementation on the Android device takes about 2 to 3 seconds longer than the runtime on MATLAB with similar images. The difference is mainly due to the delay for data transfer and starting MATLAB on the server side after each query. A simple script with minimal computation time was
written in place of the actual algorithm to measure the runtime overhead of the server-client interaction.

While, in theory, the addition of Android device should not alter the accuracy of the algorithm since the test images are all taken with the same device, the image capturing code used in this project differs from the default camera application. Although the accuracy with test images taken with default camera application is rather good, it is difficult to reproduce similar results with the custom image capturing program. As seen in figure 9, there is some difference between the images taken with the two methods.

Figure 16: Comparison between the implementation used in the project (left) and the default camera (right)

The most notable difference between the two images is the color. The image capture with custom program appears to be brighter. As color is used extensively as a method for thresholding for background removal and penny identifications, the accuracy of the algorithm is very sensitive to the color scheme of the image. The algorithm was not able to identify the pennies correctly once the lighting condition changes drastically. However, the accuracy can be restored with proper tuning of the threshold conditions. Figure 10 shows a successful attempt after modification on the threshold in MATLAB.

Figure 17: Coin recognition program running on Android mobile device

Despite the promising results already obtained, there is significant scope for additional work in order to make this Android application a practically useful one for a human user. The most obvious improvement that needs to be made is an improvement in the runtime of the algorithm. The simplest way to decrease the runtime is to reduce the number of radii values that is swept over when computing the Hough transform to create the radius image. This can be done by limiting both the range of radii (difference between maximum and minimum radii) and the resolution between consecutive radii values that are used. One way to do this might be to first construct a radius image very quickly using very coarse sampling over the entire range of radius values in order to get an estimate of the radius of each coin. Then in a second pass, finer sampling of the radii could be performed over only those radii values that are close to the estimated values. Alternatively, coarse sampling could be used simply as a way to identify the centroid of each coin (and thus separate overlapping coins) and then a quick estimation of the radius through a means such as the length of the major axis of the corresponding shape could be used. Experimental analysis would be needed to determine the specifics of the parameters necessary in the implementation, but a significant speed up in runtime without significant loss of accuracy seems probable.

Once the issue of runtime is resolved, a native solution on Android with OpenCV would become practical and should be pursued. The implementation of the algorithm based entirely on Android will cut down the overhead caused by the sever-client communication, which will further improve the performance. In addition, there are more attractive reasons to implement the algorithm in OpenCV, as the camera would be accessible while processing the image. Given that the algorithm is optimized to run fast and able to determine the threshold for background removal or penny identification in real time, it is possible for either the user or the program to adjust the camera setting to take additional image as needed if the previous threshold is not distinct enough. Furthermore, the touch screen can be used as an interface to train the program to adjust the threshold values as the user manually taps the coins that has not been identified correctly.

Even with the current setup, some immediate improvement in accuracy is possible. One major factor that hinders the accuracy is the change of lighting conditions as it is hard to come up with a good algorithm that accounts for all the different settings. Assuming that the application is to be used in a fixed location for a reasonable amount of time with mostly time-invariant lighting, it is plausible to have different sets of threshold criteria for situations such as outdoor/indoor setting similar to what is commonly implemented on a digital camera.

An additional limitation of the current implementation that could be addressed in future work is the fact that the camera must be parallel with the plane of the coins to prevent variations in the radius of identical coins across the image. Removing this constraint is conceivable and would involve applying a transformation to predict and correct for the angle of the camera with respect to the coin plane. The mathematics involved in relating a 2D projection to a 3D configuration is well
established and making this correction should be achievable with some effort.

A final improvement to be pursued would be to improve the thresholding procedure so that it works with any arbitrary background, which would make the application more feasible in a real-world situation. A more advanced technique such as locally adaptive thresholding would be necessary if the appearance of the background is varying over the image.

DIVISION OF LABOR

Mihir Pendse did 75% of the image processing implementation on MATLAB. Yiwei Wang did 25% of the image processing implementation on MATLAB and all of the Android implementation. Both authors contributed ideas to development of the algorithms and helped with compiling the poster and report.

ACKNOWLEDGEMENT

The authors would like to thank their mentor on the project, Maryam Daneshi; the teaching assistants of EE 368, David Chen and Derek Pang; and Professor Bernd Girod for their advice and instruction.

REFERENCES