

Generalized Edge Detection for Currency Value of Coins

Embedded Digital Signal Processing Laboratory
ECE420

Friday, 3:00pm



Omar Ayala (obayala2)
Ignacio Diez de Rivera (id4)
Wyatt McAllister (wmcalli2)

1.0 INTRODUCTION

1.1: Project Overview

Edge detection is a well documented topic in image processing which allows for detection of bounded shapes in complex images via contrast analysis. Our laboratory for this embedded digital signal processing class has covered basic image processing techniques as well as their implementation on the android ecosystem, and our foundational laboratory course in signal processing (ECE311) has covered edge detection in detail. For this project we will use the generalized Hough Transformation as discussed in Ballard, Dana H. “Generalizing the Hough transform to detect arbitrary shapes.”¹ to detect circles in an image captured with the android camera in order to detect coins and their relative sizes. We will take a count of the number of coins in an image and their type and use this to generate a total value in the currency selected by the user. For this project we will use dollars and Mexican pesos and detect nickels, dimes, quarters, and one, two, five, and ten peso coins.

1.2: Motivation

International travelers face challenges with currency on a daily basis. No real time algorithm for currency conversion of coins based on edge detection exists. Challenges preventing this are the need to calibrate the image based on observer distance and viewing angle. With conventional methods, it is necessary that our image be taken at a viewing angle that is normal to the plane on which the coins reside in order to accurately measure the size of the coins. However, our proposed algorithm will use the generalized Hough Transform to detect the coins as circles, alongside an accurate description of the relative sizes of all of the coins. This will allow us to obtain accurate results from a normal viewing angle, as well as obtain an accurate representation of the coins regardless of distance, as the relative sizes of US and Mexican coins have a large enough margin of error to ensure accuracy.

1.3: Benefits

- Portable implementation on android architecture allowing the application to be used with a variety of devices.
- Robust algorithm based on the generalized Hough Transform for the detection of circles, allowing coins to be detected at a variety of distances and orientations.
- Size robust size detection algorithm allowing for accurate count of different coins in a given currency based on their relative sizes.

1.4: Features

- Detects an arbitrary number of quarters, nickels and dimes, as well as one, two, five, and ten Mexican peso coins from a given image taken with the android camera.
- Displays to the user an accurate count of the total currency present, as well as the quantity of each type of coin.

2.0: LITERATURE REVIEW

INTRODUCTION

An object can be defined by its boundaries and this provides fundamental information for multiple purposes. The aim of this paper is to implement an edge detection algorithm using the Hough transformation. The Hough transformation is an algorithm for edge detection which models changes in local grey levels as a ramp function. This algorithm uses information to define a mapping from the orientation of an edge point to a reference point of the shape. The Hough transform is described in particular for analytic curves such as ellipses due to their importance in image recognition. This algorithm is generalized to arbitrary shapes which are composed of sub shapes having well defined and easily computed Hough transformations.

SUMMARY

This paper starts by analyzing the Hough transform for analytic curves. It starts with the example of a circle and details the means of extracting the directional information from this shape and using it to perform the Hough transform, as well as how to compensate for errors in the transformation. The paper next details the example of an ellipse. Since ellipses are such a common feature in everyday images, these shapes can be used to analyze images which contain complex objects. The method of performing the Hough transform for ellipses is explained, as well as its implementation and the trade offs involved in the realization of its parameter space.

The Hough transform is then generalized to non analytic curves and provides a means of computing the optimal set of parameters for a shape from its edge pixel data. This paper details earlier work on arbitrary shapes in binary edge images and then describes a means of computing the R tables for arbitrary shapes. Next this paper details several example shapes with fixed orientation and their R tables as well as the generalization of these R table properties to shapes with variable orientation. This paper next details how to use pairs of edges, find transforms for composite shapes and build convolution templates. Finally, this paper speaks about incremental strategies for building shapes using arrays of sub shapes. Methods of using locally consistent information to weight sub shapes and more complex strategies for intelligent weighting and sub shape composition are discussed.

CONCLUSION

This research details a realization of the Hough transform, allowing the detection of arbitrary shapes in an image in a manner that is invariant to scale changes, rotations, figure ground, reversals, and reference point translations. This algorithm uses the boundary of the shape to construct its R table in a manner which requires a comparable number of operations to the number of boundary points on the shape. This paper also describes a method for constructing the R table for a shape composed of sub shapes from the R tables for each sub shape. This method allows a shape to be expediently detected by means of detecting sequential sub shapes until a desired confidence level for detection has been reached. The paper also describes a means of weighting the accumulator tables for shapes in terms of locally consistent information in order to weight sub shapes in importance for recognition. Finally, the generalized Hough transform algorithms presented in this paper are parallel algorithms, allowing for faster computation time. Future work will focus on the characterization of this algorithm's computational efficiency and its feasibility as a model for object perception by biological entities.

3.0: DESIGN

3.1: Block Diagram

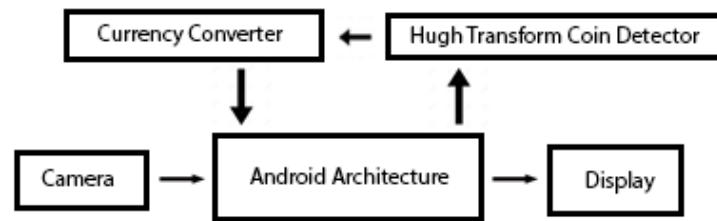


Figure 1: High Level Block Diagram

3.2: Block Descriptions

Android Architecture

- INPUTS: Camera of android device.
- DESCRIPTION: This block consists of the skeleton project files that were available to us in Lab 5 for parsing images from the camera of the Google Nexus 7 tablet, storing the image data in a buffer in YUV format, and displaying output data on the android device. We will then be able to apply the Hough Transform and coin detection algorithms directly to this data and use Eclipse and native C code to build our project.
- OUTPUTS: Display of android device.

Hough Transform Coin Detector

- INPUTS: Image output from android architecture
- PURPOSE: This block will take the image data from the buffer and apply the Hough transform to the Y channel in order to detect coins.
- OUTPUTS: Number of coins detected and their relative sizes to currency converter.

Currency Converter

- INPUTS: Transformed image with coins detected
- DESCRIPTION: This block will take the Hough Transform of the image with the detected coins, calculate the relative sizes between the coins in order to find the total number of each type of coin, and output the total currency to the monitor. Size detection requires the use of a ratio per pixel methodology which compares the relative sizes of the coins to standard size ratios found with a digital caliper. Once the various coins have been identified as either penny, nickel, dime, and quarter, one, two, five or ten-peso coin, calculating the currency value of all coins in the image requires a simple algebraic calculation with the current currency exchange.
- OUTPUTS: Currency total to display

3.3: Design Considerations



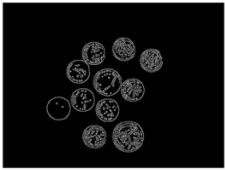


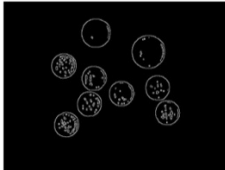


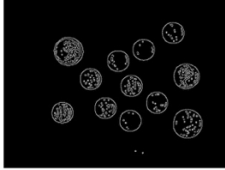

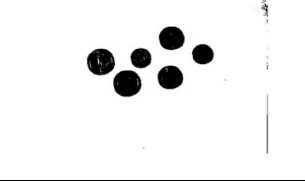
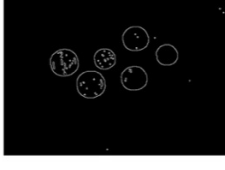

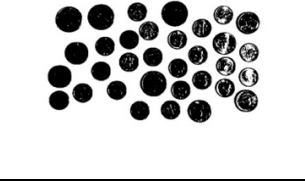
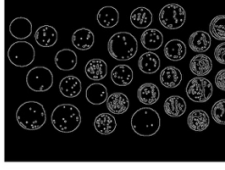
We created an application which could detect quarters, nickels, and dimes within an image containing at least one quarter or ten-peso coin. We implemented US and Mexican Currency. We neglected to detect pennies because larger numbers of pennies are needed to produce a significant currency value. For the purposes of this lab, we wanted to test our application on images with a small number of coins taking up a larger portion of the image in order to ensure high accuracy, so including pennies to start with was not a good design choice. Also, the ratio of the radius of the penny to that of the dime is so small that feature detection would need to be used, adding computation time to our algorithm and limiting its portability for mobile devices. Finally, we assumed a quarter or ten-peso coin would be present because this is the most common currency unit, and this assumption allowed us to achieve higher accuracy. Since the ratio of the radius of the quarter to that of the nickel is similar to that of the nickel to that of the dime, including the quarter allowed us to accurately detect coins without the high granularity in the radius needed if the dime is in fact the largest coin, and without the extra computation needed to check every combination of all of the radii. This increased our runtime and the portability of our application.

4.0: RESULTS

4.1: MATLAB Software

The first stage of our design process was to prototype our function in MATLAB for US currency. We wrote a script which first computed the canny edge image and called ‘imfindcircles’, an existing MATLAB library for circle detection. Our algorithm computes the the maximum radius, which we assume to be the quarter, and finds the ratio of all the other radii to this radius. We found experimentally that the ratio of the radius of the nickel to that of the quarter is less than 0.94, and that the ratio of the radius of the dime to that of the quarter is less than 0.82. These are used as thresholds to increment the counters for quarters, nickels, and dimes, which are weighted appropriately to give the total currency value. Our program gave accurate results for over several trials as detailed in **Table 1** below. This software was extended to function analogously for Mexican pesos. The code for both of these implementations is provided in **Appendix A**.

Table 1: MATLAB Coin Detection Examples for USD

RGB	Black and White	Canny	N	D	Q	T	\$
			1	9	1	11	1.2
			1	7	1	9	1
			1	9	2	12	1.45
			2	2	2	6	0.8
			6	23	4	33	1.6

4.2: C Software

This program implements our MATLAB algorithm using Open Computer Vision on an Android platform. Our algorithm first uses a Gaussian blur function from the Open Computer Vision library to reduce the noise in the image, and then passes the image into a function ‘HoughCircles’, which is analogous to ‘imfindcircles’ from MATLAB. The parameters of these function were chosen according to the design considerations detailed in **Table 2** and **Table 3** below. This function returns a vector found circles which contains the radii and centers of the circles detected. The ratio of the radii to that of the largest radius is found as before and used to compute the currency value represented in the image with the same thresholds found via our MATLAB prototyping process. Finally, the circle centers and radii are used to draw them in the image for display to the user with the currency value. As mentioned above, we extended our software implementation to work for both US dollars and Mexican Pesos. The results for several examples are shown in **Table 4** and **Table 5** below, showing the high accuracy of this program. The code for our algorithms for US and Mexican coin detection is provided in **Appendix B** and the code for the software interface with the android platform is provided in **Appendix C**.

Table 2: Design Considerations for Gaussian Blur

Function: void **GaussianBlur**(InputArray **src**, OutputArray **dst**, Size **ksize**, double **sigmaX**, double **sigmaY**=0, int **borderType**=BORDER_DEFAULT²)

Parameter	Description	Value	Design Considerations
src	Input image	src_gray	This is the input image buffer from the Android camera is used as input to the circle detection.
dst	Output Image	src_gray	Storing the output back into the same buffer conserves memory and increases runtime.
ksize	Kernel Size	Size(9,9)	This is the size of the kernel used in the Gaussian blur function. We have chosen 9 by 9, a standard size for this function.
sigmaX	Kernal Standard Deviation X	2	The standard deviation in X is 2 which is common.
sigmaY	Kernal Standard Deviation Y	2	The standard deviation in Y is 2 which is also common.

Table 3: Design Considerations for Hough Circles

Function: void **HoughCircles**(InputArray **image**, OutputArray **circles**, int **method**, double **dp**, double **minDist**, double **param1**=100, double **param2**=100, int **minRadius**=0, int **maxRadius**=0)³

Parameter		Value	Design Considerations
image	Input image	src_gray	This is the input image buffer from the Android camera which is used as input to the circle detection algorithm.
circles	Vector of circles	circles	This is the vector circles, which stores an array of circle radii and centers.
method	Detection method	CV_HOUGH_GRADIENT	This is the method specified for circle detection, currently the only one available.
dp	Ratio of Image to Accumulator Resolution	1	The ratio of image to accumulator resolution is chosen to be unity. Higher accumulator resolution provides greater accuracy as circles with higher accumulator thresholds are flagged with higher accuracy for display. The unity ratio provides the largest possible accumulator resolution and thus the largest granularity to flag circle accuracy.
minDist	Minimum distance between circles	50	This parameter specifies the minimum distance between circles necessary for accurate detection. In our case, this parameter ensures that invalid circles, such as concentric circles within coins, are not detected.
Param1	Higher threshold of canny edges	255	This is the larger threshold for canny edges, specifying the maximum intensity for pixel values, in our case 255.
Param2	Accumulator detection threshold	30	This is the accumulator threshold necessary for a circle to be flagged as valid for display.
minRadius	Minimum radius	10	This is the minimum circle radius necessary for a valid circle to be displayed. Setting this parameter to 10 prevents spurious circles from being detection, while still allowing coins to be detection in images taken from larger distance.
maxRadius	Maximum circle radius	200	This is the maximum radius for circle detection. This parameter to prevents spurious circles from being detection on the around the boundary of a group of closely spaced coins, while still allowing coins to be detected in images taken from a smaller distance.

Table 4: OpenCV Coin Detection Examples for USD


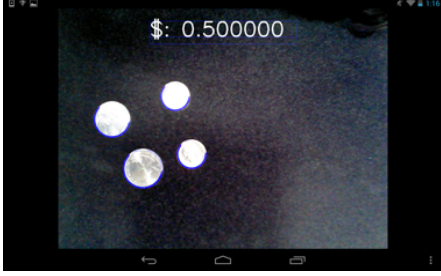





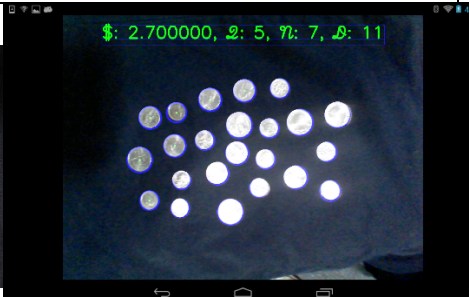








RGB	Black and White	N	D	Q	T	\$
		1	2	1	4	0.5
		2	2	2	6	0.8
		1	0	2	0	0.45
		5	7	11	23	2.7

Table 5: OpenCV Coin Detection Examples for Mexican Pesos

RGB	Black and White	U	D o	C	D i	\$
		1	0	3	1	26
		2	1	1	1	19
		3	3	3	1	34
		3	3	3	2	44

5.0: SUGGESTIONS FOR EXTENSIONS AND MODIFICATIONS

5.1: Future Goals

Our coin program can easily be extended to function for a number of user specified currencies. A database would be implemented which would store the relative radii of coins in other currency systems and a simple case would be used to switch the thresholds to those appropriate for the chosen currency. This program provides a proof of concept for extension to an application capable of facilitating international currency exchange. For specific currency systems, feature detection and relative color intensity could be used to distinguish between coins which are of the same size. Finally, histogram equalization could be used to improve contrast in low light conditions and ellipse detection could be implemented on faster systems to enable high accuracy for images taken at an angle. This would enable the creation of automated systems which use a gantry and a real time camera to count large numbers of coins before sorting.

APPENDIX A: MATLAB SOFTWARE

A.1: US Dollars

```
clear, clc                                     %Clear workspace and command window
q=0; n=0; d=0; level=0.55;                   %Initialize variables
A = imread('monedas11.JPG');                 %Read in RGB image
figure; imshow(A)                            %Display RGB Image
I = im2bw(A, level);                         %Compute Black and White Image
figure; imshow(I)                           %Display Black and White Image
I = edge(I, 'canny');                        %Compute Canny Edge Image
figure; imshow(I)                            %Display Canny Edge Image

[centers1, radii1, metric1] = imfindcircles(I, [7 21]); %Detect Circles1
[centers2, radii2, metric2] = imfindcircles(I, [22 66]); %Detect Circles2
radii=[radii1; radii2];                      %Vector of radii

maxrad=max(radii);                           %Find maximum radius
relradius=radii/max(radii);                  %Compute the vector of ratio radii
i=1;                                         %Initialize I to zero
while i<length(relradius)+1                 %Loop through radius array
    if relradius(i)<0.82                     %For all radii less than the dime
        d=d+1;                               %Increment the number of dimes
    elseif relradius(i)<0.94                 %For all radii less than the nickel
        n=n+1;                               %Increment number of nickels
    elseif relradius(i)<1.01                 %For all radii less than the quarter
        q=q+1;                               %Increment number of Quarters
    end                                       %End if
    i=i+1;                                   %Increment i
end

n, d, q                                     %total nickels, dimes, and quarters
total_coins=q+n+d                           %total coins
money=(q*25+d*10+n*5)/100                   %total money ($)
```

A.2: MEX Pesos

```
clear, clc                                     %Clear workspace and command window
uno=0; dos=0; cinco=0; diez=0; level=0.55; %Initialize variables
A = imread('monedas24.JPG');                 %Read in RGB image
figure; imshow(A)                            %Display RGB Image
I = im2bw(A, level);                         %Compute Black and White Image
figure; imshow(I)                           %Display Black and White Image
I = edge(I, 'canny');                        %Compute Canny Edge Image
figure; imshow(I)                            %Display Canny Edge Image

[centers1, radii1, metric1] = imfindcircles(I, [7 21]); %Detect Circles1
[centers2, radii2, metric2] = imfindcircles(I, [22 66]); %Detect Circles2
radii=[radii1; radii2];                      %Vector of radii

maxrad=max(radii);                           %Find maximum radius
relradius=radii/max(radii);                  %Compute the vector of ratio radii
i=1;                                         %Initialize I to zero
while i<length(relradius)+1                 %Loop through radius array
    if relradius(i)<0.805                    %For all radii less than the uno
        uno=uno+1;                          %Increment the number of uno
    elseif relradius(i)<0.89                 %For all radii less than the dos
        dos=dos+1;                          %Increment number of dos
    elseif relradius(i)<0.96                 %For all radii less than the cinco
        cinco=cinco+1;                      %Increment number of cinco
    elseif relradius(i)<1.01                 %For all radii less than the diez
        diez=diez+1;                        %Increment number of diez
    end                                       %End if
    i=i+1;                                   %Increment i
end

uno, dos, cinco, diez                       %total unos, doses, cincos, dieces
total_coins= uno+dos+cinco+diez             %total coins
money = uno+dos*2+cinco*5+diez*10           %total money ($)
```

APPENDIX B: C SOFTWARE CODE

```
#include <jni.h>
#include <opencv2/core/core.hpp>
#include <android/log.h>

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>
#include <string.h>

using namespace std;
using namespace cv;

extern "C" {

// function declaration for manual color conversion
int convertYUVtoARGB(int, int, int);

JNIEXPORT void JNICALL Java_org_ece420_lab5_Sample4View_YUV2RGB(JNIEnv*, jobject, jlong addrYuv, jlong
addrRgba)
{
    Mat* pYUV=(Mat*)addrYuv;
    Mat* pRGB=(Mat*)addrRgba;

    /*INSERT CODE TO CONVERT AN ENTIRE IMAGE FROM FROM YUV420sp TO ARGB*/
    int width = pYUV->cols;
    int height = (pYUV->rows)*2/3;
    int u, v, y1, y2, y3, y4;
    int index = 0;

    for(int j=0; j<height; j+=2)
    {
        for(int i=0; i<width; i+=2)
        {
            y1 = pYUV->at<uchar>(j,i)&0xff;
            y2 = pYUV->at<uchar>(j,i+1)&0xff;
            y3 = pYUV->at<uchar>(j+1,i)&0xff;
            y4 = pYUV->at<uchar>(j+1,i+1)&0xff;

            u = pYUV->at<uchar>(index/width+height,index*width)&0xff;
            v = pYUV->at<uchar>(index/width+height,index*width+1)&0xff;

            pRGB->at<int>(j,i) = convertYUVtoARGB(y1, u, v);
            pRGB->at<int>(j,i+1) = convertYUVtoARGB(y2, u, v);
            pRGB->at<int>(j+1,i) = convertYUVtoARGB(y3, u, v);
            pRGB->at<int>(j+1,i+1) = convertYUVtoARGB(y4, u, v);

            index+=2;
        }
    }
}

JNIEXPORT double JNICALL Java_org_ece420_lab5_Sample4View_HistEQ(JNIEnv* env, jobject thiz, jlong addrYuv,
jlong addrRgba)
{
    Mat* pYUV=(Mat*)addrYuv;
    Mat* pRGB=(Mat*)addrRgba;

    Mat src_gray;

    int d=0;
    int n=0;
    int q=0;
    int total_coins=0;
    double currency=0;
    int threshold_value=50;
```

```

    int Max_Binary_Value=100;

    vector<Vec3f> circles;

// convert to rgb and store
Java_org_ece420_lab5_Sample4View_YUV2RGB(env, this, addrYuv, addrRgba);

    // Convert it to gray
    cvtColor(*pRGB, src_gray, CV_BGR2GRAY );

    //convert it to binary
    //threshold(src_gray,src_gray,threshold_value,Max_Binary_Value,0);

    GaussianBlur(src_gray,src_gray,Size(9,9),2,2);

    // Apply the Hough Transform to find the circles
    HoughCircles(src_gray, circles, CV_HOUGH_GRADIENT, 1, 50, 255, 30, 10, 200);
    double radius[(int) circles.size()];
    double relradius[(int) circles.size()];
    double max_radius;
max_radius=0;

    for(size_t i = 0; i < circles.size(); i++ )
    {
        radius[i] = cvRound(circles[i][2]);
        if(radius[i]>max_radius)
        {
            max_radius=radius[i];
        }
    }

    for(size_t i = 0; i < circles.size(); i++ )
    {
        relradius[i]=radius[i]/max_radius;
        // Draw the circles detected
        Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
        // circle center
        circle(*pRGB, center, 3, Scalar(0,255,0), -1, 8, 0);
        // circle outline
        circle(*pRGB, center, radius[i], Scalar(0,0,255), 3, 8, 0);

        if (relradius[i] < 0.82){
            d++;
        }
        else if (relradius[i] < 0.94){
            n++;
        }
        else if (relradius[i] < 1.01){
            q++;
        }
    }

    }

total_coins=q+n+d;
currency=(q*25.0+d*10.0+n*5.0)/100.0;
char text[200];
//sprintf(text,"Size: %d %d",relradius[0], relradius[1]);
sprintf(text,"$: %f, Q: %d, N: %d, D: %d",currency,q,n,d);
int fontFace = FONT_HERSHEY_DUPLEX;
double fontScale = 1.5;
int thickness = 3;

int baseline=0;
Size textSize = getTextSize(text, fontFace,
                            fontScale, thickness, &baseline);
baseline += thickness;

// center the text
Point textOrg((src_gray.cols - textSize.width)/2,
              (textSize.height)*2);

```

```

// draw the box
rectangle(*pRGB, textOrg + Point(0, baseline),
          textOrg + Point(textSize.width, -textSize.height),
          Scalar(0,0,255));
// ... and the baseline first
line(*pRGB, textOrg + Point(0, thickness),
     textOrg + Point(textSize.width, thickness),
     Scalar(0, 0, 255));

// then put the text itself
putText(*pRGB, text, textOrg, fontFace, fontScale,
        Scalar(0, 0, 180), thickness, 8);

return currency;
}

JNIEXPORT double JNICALL Java_org_ece420_lab5_Sample4View_HistEQ1(JNIEnv* env, jobject thiz, jlong
addrYuv, jlong addrRgba)
{
    Mat* pYUV=(Mat*)addrYuv;
    Mat* pRGB=(Mat*)addrRgba;

    Mat src_gray;

    int uno=0;
    int dos=0;
    int cinco=0;
    int diez=0;
    int total_coins=0;
    int currency=0;
    int threshold_value=50;
    int Max_Binary_Value=100;

    vector<Vec3f> circles;

    // convert to rgb and store
    Java_org_ece420_lab5_Sample4View_YUV2RGB(env, thiz, addrYuv, addrRgba);

    // Convert it to gray
    cvtColor(*pRGB, src_gray, CV_BGR2GRAY );

    //convert it to binary
    //threshold(src_gray,src_gray,threshold_value,Max_Binary_Value,0);

    GaussianBlur(src_gray,src_gray,Size(9,9),2,2);

    // Apply the Hough Transform to find the circles
    HoughCircles(src_gray, circles, CV_HOUGH_GRADIENT, 1, 50, 255, 30, 10, 200);
    double radius[(int) circles.size()];
    double relradius[(int) circles.size()];
    double max_radius;
    max_radius=0;

    for(size_t i = 0; i < circles.size(); i++ )
    {
        radius[i] = cvRound(circles[i][2]);
        if(radius[i]>max_radius)
        {
            max_radius=radius[i];
        }
    }

    for(size_t i = 0; i < circles.size(); i++ )
    {
        relradius[i]=radius[i]/max_radius;
        // Draw the circles detected
        Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
        // circle center

```



```

        circle(*pRGB, center, 3, Scalar(0,255,0), -1, 8, 0);
        // circle outline
        circle(*pRGB, center, radius[i], Scalar(0,0,255), 3, 8, 0);

        if (relradius[i] < 0.80){
            uno++;
        }
        else if (relradius[i] < 0.87){
            dos++;
        }
        else if (relradius[i] < 0.965){
            cinco++;
        }
        else if (relradius[i] < 1.01){
            diez++;
        }
    }

    total_coins=uno+dos+cinco+diez;
    currency=uno+dos*2+cinco*5+diez*10;
    char text[200];
    //sprintf(text,"Size: %d %d",relradius[0], relradius[1]);
    sprintf(text,"$: %d, U: %d, Do: %d, C: %d, Di: %d ",currency, uno, dos, cinco, diez);
    int fontFace = FONT_HERSHEY_DUPLEX;
    double fontScale = 1.5;
    int thickness = 3;

    int baseline=0;
    Size textSize = getTextSize(text, fontFace,
                                fontScale, thickness, &baseline);
    baseline += thickness;

    // center the text
    Point textOrg((src_gray.cols - textSize.width)/2,
                  (textSize.height)*2);

    // draw the box
    rectangle(*pRGB, textOrg + Point(0, baseline),
              textOrg + Point(textSize.width, -textSize.height),
              Scalar(0,0,255));
    // ... and the baseline first
    line(*pRGB, textOrg + Point(0, thickness),
         textOrg + Point(textSize.width, thickness),
         Scalar(0, 0, 255));

    // then put the text itself
    putText(*pRGB, text, textOrg, fontFace, fontScale,
            Scalar(180, 0, 0), thickness, 8);

    return currency;
}

int convertYUVtoARGB(int y, int u, int v) {
    /*INSERT CODE TO CONVERT A YUV PIXEL TO A 32-BIT INT REPRESENTING AN ARGB PIXEL*/
    int r,g,b;

    r = y + (int)(1.370705 * (v-128));
    g = y - (int)(0.698001 * (v-128)) - (0.337633 * (u-128));
    b = y + (int)(1.732446 * (u-128));

    r = r > 255 ? 255 : r < 0 ? 0 : r;
    g = g > 255 ? 255 : g < 0 ? 0 : g;
    b = b > 255 ? 255 : b < 0 ? 0 : b;

    return 0xff000000 | (r<<16) | (g<<8) | b;
}

```


APPENDIX C: JAVA CODE

C.A: SAMPLE4VIEW

```
package org.ece420.lab5;

import org.opencv.android.Utils;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.imgproc.Imgproc;

import android.content.Context;
import android.graphics.Bitmap;
import android.util.Log;

class Sample4View extends SampleViewBase {
    private static final String TAG = "OCVSample:View";

    public static final int VIEW_MODE_GRAY = 0;
    public static final int VIEW_MODE_RGBA = 1;
    public static final int VIEW_MODE_HISTEQ = 2;

    private Mat mYuv;
    private Mat mRgba;
    private Mat mGraySubmat;
    private Bitmap mBitmap;
    private int mViewMode;
    private int first;

    public Sample4View(Context context) {
        super(context);
    }

    @Override
    protected void onPreviewStarted(int previewWidth, int previewHeight) {
        Log.i(TAG, "called onPreviewStarted("+previewWidth+", "+previewHeight+)");

        // initialize Mats before usage
        mYuv = new Mat(getFrameHeight() + getFrameHeight() / 2, getFrameWidth(), CvType.CV_8UC1);
        mGraySubmat = mYuv.submat(0, getFrameHeight(), 0, getFrameWidth());

        // allocate space now because are using our own color conversion function
        mRgba = new Mat(getFrameHeight(), getFrameWidth(), CvType.CV_8UC4);

        mBitmap = Bitmap.createBitmap(previewWidth, previewHeight, Bitmap.Config.ARGB_8888);
    }

    @Override
    protected void onPreviewStopped() {
        Log.i(TAG, "called onPreviewStopped");

        if (mBitmap != null) {
            mBitmap.recycle();
            mBitmap = null;
        }

        synchronized (this) {
            // Explicitly deallocate Mats
            if (mYuv != null)
                mYuv.release();
            if (mRgba != null)
                mRgba.release();
            if (mGraySubmat != null)
                mGraySubmat.release();

            mYuv = null;
            mRgba = null;
            mGraySubmat = null;
        }
    }
}
```

```

@Override
protected Bitmap processFrame(byte[] data) {
    // data from camera is in YUV420sp format
    mYuv.put(0, 0, data);

    final int viewMode = mViewMode;

    if(viewMode==VIEW_MODE_GRAY){
        // opencv's color conversion function
        Imgproc.cvtColor(mGraySubmat, mRgba, Imgproc.COLOR_GRAY2RGBA, 4);
        first=1;
    }
    if(viewMode==VIEW_MODE_RGBA){
        // apply equalization to Y channel and convert to RGB
        if(first==1){
            double currency;
            currency=HistEQ1(mYuv.getNativeObjAddr(), mRgba.getNativeObjAddr());
            first=0;
        }
    }
    if(viewMode==VIEW_MODE_HISTEQ){
        // apply equalization to Y channel and convert to RGB
        if(first==1){
            double currency;
            currency=HistEQ(mYuv.getNativeObjAddr(), mRgba.getNativeObjAddr());
            first=0;
        }
    }
    Bitmap bmp = mBitmap;

    try {
        Utils.matToBitmap(mRgba, bmp);
    } catch(Exception e) {
        Log.e("org.opencv.samples.puzzle15", "Utils.matToBitmap() throws an exception: " +
            e.getMessage());
        bmp.recycle();
        bmp = null;
    }

    return bmp;
}

public native void YUV2RGB(long matAddrYUV, long matAddrRgba);
public native double HistEQ(long matAddrYUV, long matAddrRgba);
public native double HistEQ1(long matAddrYUV, long matAddrRgba);

public void setViewMode(int viewMode) {
    Log.i(TAG, "called setViewMode("+viewMode+)");
    mViewMode = viewMode;
}
}

```

C.2: Sample4Mixed

```
package org.ece420.lab5;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.Window;
import android.view.WindowManager;

public class Sample4Mixed extends Activity {
    private static final String TAG = "OCVSample::Activity";

    private MenuItem mItemPreviewRGBA;
    private MenuItem mItemPreviewGray;
    private MenuItem mItemPreviewHistEq;
    private Sample4View mView;

    private BaseLoaderCallback mOpenCVCallBack = new BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
            switch (status) {
                case LoaderCallbackInterface.SUCCESS:
                {
                    Log.i(TAG, "OpenCV loaded successfully");

                    // Load native library after(!) OpenCV initialization
                    System.loadLibrary("mixed_sample");

                    // Create and set View
                    mView = new Sample4View(mAppContext);
                    setContentView(mView);

                    // Check native OpenCV camera
                    if( !mView.openCamera() ) {
                        AlertDialog ad = new AlertDialog.Builder(mAppContext).create();
                        ad.setCancelable(false); // This blocks the 'BACK' button
                        ad.setMessage("Fatal error: can't open camera!");
                        ad.setButton(AlertDialog.BUTTON_POSITIVE, "OK", new
                            DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog, int which) {
                                    dialog.dismiss();
                                    finish();
                                }
                            });
                        ad.show();
                    }
                }
                } break;

                /** OpenCV loader cannot start Google Play */
                case LoaderCallbackInterface.MARKET_ERROR:
                {
                    Log.d(TAG, "Google Play service is not accessible!");
                    AlertDialog MarketErrorMessage = new
                        AlertDialog.Builder(mAppContext).create();
                    MarketErrorMessage.setTitle("OpenCV Manager");
                    MarketErrorMessage.setMessage("Google Play service is not accessible!\nTry to
                        install the 'OpenCV Manager' and the appropriate 'OpenCV binary
                        pack' APKs from OpenCV SDK manually via 'adb install' command.");
                    MarketErrorMessage.setCancelable(false); // This blocks the 'BACK' button
                    MarketErrorMessage.setButton(AlertDialog.BUTTON_POSITIVE, "OK", new
                        DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int which) {

```

```

        ((Activity) mAppContext).finish();
    }
    });
    MarketErrorMessage.show();
} break;
default:
{
    super.onManagerConnected(status);
} break;
}
}
};

public Sample4Mixed() {
    Log.i(TAG, "Instantiated new " + this.getClass());
}

@Override
protected void onPause() {
    Log.i(TAG, "called onPause");
    if (null != mView)
        mView.releaseCamera();
    super.onPause();
}

@Override
protected void onResume() {
    Log.i(TAG, "called onResume");
    super.onResume();

    Log.i(TAG, "Trying to load OpenCV library");
    if (!OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_2, this, mOpenCVCallBack)) {
        Log.e(TAG, "Cannot connect to OpenCV Manager");
    }
}

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    Log.i(TAG, "called onCreate");
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    Log.i(TAG, "called onCreateOptionsMenu");
    mItemPreviewGray = menu.add("Take Photo");
    mItemPreviewRGBA = menu.add("Apply Coin Detection Mexican Pesos");
    mItemPreviewHistEq = menu.add("Apply Coin Detection USD");
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    Log.i(TAG, "called onOptionsItemSelected; selected item: " + item);
    if (item == mItemPreviewRGBA) {
        mView.setViewMode(Sample4View.VIEW_MODE_RGBA);
    } else if (item == mItemPreviewGray) {
        mView.setViewMode(Sample4View.VIEW_MODE_GRAY);
    } else if (item == mItemPreviewHistEq) {
        mView.setViewMode(Sample4View.VIEW_MODE_HISTEQ);
    }
    return true;
}
}
}
}

```

6.0: WORKS CITED

[1] Ballard, Dana H. "Generalizing the Hough transform to detect arbitrary shapes." *Pattern recognition* 13.2 (1981): 111-122.

[2] "Image Filtering — OpenCV 2.4.13.1 documentation", *Docs.opencv.org*, 2016. [Online]. Available: <http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblur>. [Accessed: 23- Nov- 2016].

[3] "Feature Detection — OpenCV 2.4.13.1 documentation", *Docs.opencv.org*, 2016. [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles. [Accessed: 23- Nov- 2016].