**Jared Hayes**

**CPE 403 Advanced Embedded Systems Section 1001**

**OpenCV Detection of Various Denominations of Coins**

December 3, 2014

**Report\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Objective

* Cross-compile on BeagleBone Black to create C++ application
* Utilize OpenCV libraries to apply edge detection to an image/video of coins
* Detect and outline edges of various coins
* Distinguish between various denominations of coins and determine monetary value

Background

OpenCV is a library of programming functions that aides in the processing and analyzing of images/videos .

Tools

* BeagleBone Black rev. c
* 16GB microSD
* Logitech HD Webcam C270
* Ubuntu 14.04
* Eclipse
* OpenCV
* Remote System Explorer End-User Runtime
* gcc/g++-arm-linux-gnueabi package

Host Environment

* Ubuntu 14.04 running on VirtualBox VM in Windows 8.1
* Utilizing Eclipse IDE to cross-compile for BeagleBone Black
  + Allows for quicker compilation and the use of an IDE
  + Must use an ARM toolchain to communicate with BBB
* Install Eclipse
  + sudo apt-get install eclipse eclipse-cdt g++ gcc
* Install Remote System Explorer End User Runtime package in Eclipse
* Install C/C++ Remote Launch plugin in Eclipse
* Install Toolchain for BBB
  + sudo apt-get install gcc-arm-linux-gnueabi
  + sudo apt-get install g++-arm-linux-gnueabi

Configure Toolchain

* Eclipse > Project Properties > C/C++ Build > Settings > Tool Settings
* Change GCC/G++ Complier, Linker, Assembler to ARM toolchain equivalents
* Library Path: /usr/arm-linux-gnueabi/lib
* Include Path: /usr/arm-linux-gnueabi/include
* Establish Remote Connection to BBB in Run Configurations for remote deployment

BeagleBone Black Environment

* Angstrom – default BBB distribution, already contains OpenCV libraries
* Download latest distribution <http://beagleboard.org/latest-images/>
* Copied image in Windows using Win32DiskImager to uSD
* Flash BBB
* SSH into BBB
  + Ssh 192.168.7.2 –l root

Install OpenCV

* Install up to date OpenCV prereqs
  + sudo apt‐get install build‐essential cmake pkg‐config
  + sudo apt‐get install libtiff4‐dev libjpeg‐dev libjasper‐dev libpng12‐dev
  + sudo apt‐get install libavcodec‐dev libavformat‐dev libswscale‐dev libv4l‐dev
* Download OpenCV from Github
  + git clone <https://github.com/Itseez/opencv.git>
* Build OpenCV
  + cd opencv
  + mkdir build && cd build
  + cmake ‐D CMAKE\_BUILD\_TYPE=RELEASE ‐D CMAKE\_INSTALL\_PREFIX=/usr/local ‐D WITH\_CUDA=OFF ‐D WITH\_CUFFT=OFF ‐D WITH\_CUBLAS=OFF ‐D WITH\_NVCUVID=OFF ‐D WITH\_OPENCL=OFF ‐D WITH\_OPENCLAMDFFT=OFF ‐D WITH\_OPENCLAMDBLAS=OFF ‐D BUILD\_opencv\_apps=OFF ‐D BUILD\_DOCS=OFF ‐D BUILD\_PERF\_TESTS=OFF ‐D BUILD\_TESTS=OFF ‐D ENABLE\_NEON=on ..
  + make
  + sudo make install
  + sudo ldconfig

Implementation

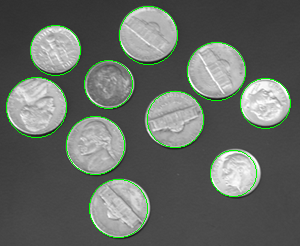
*OpenCV Functions*

* Load an image
  + src = imread( argv[1], 1 );
* Convert it to grayscale:
  + cvtColor( src, src\_gray, CV\_BGR2GRAY );
* Apply a Gaussian blur to reduce noise and avoid false circle detection:
  + GaussianBlur( src\_gray, src\_gray, Size(9, 9), 2, 2 );
* Create window
  + char\* source\_window = "Source"; namedWindow( source\_window, CV\_WINDOW\_AUTOSIZE ); imshow( source\_window, src );

The above is used to prepare the image for processing. Then, the Hough Circles function is used

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

in order to detect circle edges. The circle() function then draws circles to outline the coins in the image and then a window is opened with the new image. The minRadius and maxRadius parameters are used to filter out detection of circles out of the specified radius range. The minDist parameter can be adjusted if not all of the coins are detected or extra false coins are being detected.



*Bounded Rectangles*



The minAreaRect and fitEllipse functions can be used to find the rotated rectangle and ellipses for each contour (coin). The rectangle which binds the outlining ellipse can be used to determine the diameter of the coin.

* Detect Edges using threshold
  + threshold( src\_gray, threshold\_output, thresh, 255, THRESH\_BINARY );
* Find Contours
  + findContours( threshold\_output, contours, hierarchy, CV\_RETR\_TREE, CV\_CHAIN\_APPROX\_SIMPLE, Point(0, 0) );
* Draw contours + rotated rects + ellipses
  + ellipse( drawing, minEllipse[i], color, 2, 8 )
  + for( int j = 0; j < 4; j++ ){

line( drawing, rect\_points[j], rect\_points[(j+1)%4], color, 1, 8 );}

The rectangle is drawn with the line() function in a four loop with 4 counts. The lines are drawn according to the 4 points of the detected rectangle, which are stored in the array rect\_points[j]. Subtracting any two of the points will roughly yield the diameter of the coin, since the ellipse outlining the coin is perfectly bounded in the rectangle. And since the coin should be circular rather than elliptical, the rectangle should be square.

diam = abs(rect\_points[j].-rect\_points[j+1].

The returned diameter value is measured in pixels. This makes it difficult to translate into an actual diameter value because the number of pixels of the coin can vary depending on how the image is taken or scaled.

If the actual diameter value could be determined, then I could determine what type of coin it is and assign it a monetary value. The program would keep a running sum of the values and display the final dollar amount.

// Distinguish coins

if(diam == 10){

cout << "Quarter: $0.25" << endl;

sum = sum + .25;}

else if(diam == 5){

cout << "Nickel: $0.05" << endl;

sum = sum + .25;}

else if(diam == 3){

cout << "Dime: $0.10" << endl;

sum = sum + .25;}

else(diam == 1){

cout << "Penny: $0.01" << endl;

sum = sum + .25; }

}

cout << "Total Amount: " sum << endl;

It is only possible to set diameter values for a particular image or video capture point.

Conclusion

Hough Transform is sufficient for detecting non-overlapping coins of various sizes without false circles and missing circles. Distinguishing between different coins for all input images does not seem practical. The extension of this project would be to apply this to video capture in order to detect the coins in real time.

Code: (parts of code sampled from OpenCV examples)

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

using namespace cv;

using namespace std;

Mat src; Mat src\_gray;

int thresh = 100;

int max\_thresh = 255;

RNG rng(12345);

int diam;

int sum;

/// Function header

void thresh\_callback(int, void\* );

/\*\* @function main \*/

int main( int argc, char\*\* argv )

{

/// Load source image and convert it to gray

src = imread( argv[1], 1 );

/// Convert image to gray and blur it

cvtColor( src, src\_gray, CV\_BGR2GRAY );

blur( src\_gray, src\_gray, Size(3,3) );

/// Create Window

char\* source\_window = "Source";

namedWindow( source\_window, CV\_WINDOW\_AUTOSIZE );

imshow( source\_window, src );

createTrackbar( " Threshold:", "Source", &thresh, max\_thresh, thresh\_callback );

thresh\_callback( 0, 0 );

waitKey(0);

return(0);

}

/\*\* @function thresh\_callback \*/

void thresh\_callback(int, void\* )

{

Mat threshold\_output;

vector<vector<Point> > contours;

vector<Vec4i> hierarchy;

/// Detect edges using Threshold

threshold( src\_gray, threshold\_output, thresh, 255, THRESH\_BINARY );

/// Find contours

findContours( threshold\_output, contours, hierarchy, CV\_RETR\_TREE, CV\_CHAIN\_APPROX\_SIMPLE, Point(0, 0) );

/// Find the rotated rectangles and ellipses for each contour

vector<RotatedRect> minRect( contours.size() );

vector<RotatedRect> minEllipse( contours.size() );

for( int i = 0; i < contours.size(); i++ )

{ minRect[i] = minAreaRect( Mat(contours[i]) );

if( contours[i].size() > 5 )

{ minEllipse[i] = fitEllipse( Mat(contours[i]) ); }

}

/// Draw contours + rotated rects + ellipses

Mat drawing = Mat::zeros( threshold\_output.size(), CV\_8UC3 );

for( int i = 0; i< contours.size(); i++ )

{

Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );

// contour

drawContours( drawing, contours, i, color, 1, 8, vector<Vec4i>(), 0, Point() );

// ellipse

ellipse( drawing, minEllipse[i], color, 2, 8 );

// rotated rectangle

Point2f rect\_points[4]; minRect[i].points( rect\_points );

for( int j = 0; j < 4; j++ ){

line( drawing, rect\_points[j], rect\_points[(j+1)%4], color, 1, 8 );}

diam = abs(rect\_points[j].-rect\_points[j+1].)

// Distinguish coins

if(diam == 10)

cout << "Quarter: $0.25" << endl;

sum = sum + .25;

if(diam == 5)

cout << "Nickel: $0.05" << endl;

sum = sum + .25;

if(diam == 3)

cout << "Dime: $0.10" << endl;

sum = sum + .25;

if(diam == 11)

cout << "Penny: $0.01" << endl;

sum = sum + .25;

}

cout << "Total Amount: " sum << endl;

/// Show in a window

namedWindow( "Contours", CV\_WINDOW\_AUTOSIZE );

imshow( "Contours", drawing );

}

References

* <https://learn.adafruit.com/downloads/pdf/beaglebone-black-installing-operating-systems.pdf>
* <http://www.michaelhleonard.com/cross-compile-for-beaglebone-black/>
* <https://drive.google.com/viewerng/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnx1bmx2Y3BlNDAzfGd4OmE2NDNmN2ZhMjkyYjFlZQ>
* [http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough\_circle/hough\_circle.html#hough-circle](http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html)