Summary

`CoinDetector0` is an algorithm that combines basic image processing steps of smoothing, recoloring, exposure correction, morphological operations and edge and circle detection to identify coins in top-down images of sets of coins. The only user input needed is a path to an image file; input parameters to the processing steps have been set by heuristic evaluation on several coin data sets, and the sequence of steps is determined by the coefficient of variation (ratio of standard deviation to mean) of image intensities. CoinDetector0 also provides basic annotation functionality and reports annotated and detected circle centers.

Image Set

Images were selected to have relatively small size (~1 mega pixels), to show a small range of coin sizes, to have an easy-to-count number of coins, and to have coins photographed nearly from above so that they appear circular.

Images were sourced from image searches for "coin collections" on Bing and Google. Table 1, below, gives summary information on each coin image; Appendix II shows image thumbnails and histograms.

Image	Size	# Coins	Notes
1	701x1000	13	dark background with bright, shiny coins
2	794x529	19	text outside of coins
3	541x600	9	shiny coins and shadowy coins
4	720x750	10	bright white background after masking
5	314x800	12	bright white background after masking
6	889x1045	9	red background
7	589x1000	10	coins may be drawings
8	750x1000	9	angled camera
9	500x500	10	hexagonal coin
10	853x640	12	shadows, textured background

Table 1: Characteristics of coin images

Annotation Procedure

If an annotation plain text file does not exist for an image, `CoinDetector0` will prompt the user to annotate circles. The annotation interface is built around a mousecallback in which the user can draw circles on a greyscale copy of the image by dragging a rectangle which circumscribes the circle. The center of the circle appears as a white dot. The x, y coordinates of each circle center are stored in a plain text file `[image_root_name]true_circles.txt`, which is to be read into memory so that the algorithm-detected circle set can be evaluated.

{functions: `my_mouse_callback`, `draw_box`, `printCircle`, `CircleTexttoVec`}

Scoring the Algorithm's Performance

To compare the annotated circles to the algorithm-predicted circles, the algorithm loops over both sets of circles and computes the Euclidean distance between the center of each predicted circle and the center of each annotated circle.

A true positive (TP) is scored when the distance between algorithm-predicted circle center and one of the annotated circle centers is less than half of the minimum-distance-between-centers parameter supplied to the Hough circle transform. Only one true positive prediction is allowed per annotated circle, which is enforced by breaking out of the (inner) loop over annotated circles and removing the correctly-predicted circle from the list of annotated circles before the next iteration of the (outer) loop of algorithm-predicted circles. A false positive (FP) is scored when there is no annotated circle with half of the Hough minimum distance of an algorithm-predicted circle. The number of false negatives (FNs) is given by the number of un-matched annotated circles.

When running `CoinDetector0`, centers, radii of detected circles, in addition to TPs, FPs, FNs, and F1s are recorded in [`image_root_name`]report.txt files in the dataset.

{functions: `F1Finder`, `readF1File`}

Algorithm

Step	Purpose/Notes
Bilateral filter	Blur coin engravings to minimize number of edges
(Check for) second bilateral filter	Compute image mean/std dev, apply extra blurring to images with low variation of intensities
Greyscale	
Gamma correction	Minimize the number of shadows detected as edges
(Check for) histogram equalization	Compute image mean/std dev, equalize histogram of images with high variation of image intensities
Gaussian blur	Blur entire greyscale image before thresholding/morphology
Triangle threshold	Remove shines/highlights
Canny edge detection	Detect coin edges, taking advantage of hysteresis thresholding
Morphological opening	Remove spurious edges detected from shadows
Morphological closing	Fill in gaps in coins' perimeters
Hough circle detection	Detect circles, check for circles with outlying radii (>1.5IQR above/below median radius) and, if detected, rerun circle detection with minimum/maximum radii
(Check for) annotation	Check for existence of file annotating true coin centers; prompt user to annotate coins if it does not exist

Calculate TP, FP, FN, F1	

Table 2: Steps of the CoinDetector0 algorithm

Despite Canny Edge detection being part of the `HoughCircles` function, we apply edge finding and thresholding as separate steps before calling `HoughCircles` so that morphological transformations could be applied to the detected edges and force the perimeters of coins to be connected.

{functions: `cannyWrap`, `houghCircleWrap`, `getmiddlenumber`, `CircleTexttoVec`, `median`, `IQR`}

Algorithm Design

The algorithm was developed iteratively, starting with the bare `HoughCircles` function, with additional functionality added to process images conditionally on color characteristics.

Similarly, parameter values were set by ad-hoc tuning to maximize F1 scores. For example, for the bilateral filtering step, neighborhood diameters between 16 and 144 were considered on different sets of images and an informal binary search was used, resulting in the final neighborhood diameter value of 20 px.

Special Features

- Extra bilateral filtering, conditional on distribution of image intensities
- Histogram equalization conditional on distribution of image intensities
- Outlier cleaning of circles in order to avoid “detection” of “coins” that have radii that fall far outside the range of radii observed in the middle 50% of detected coins.
- At each major stage (smoothing, conversion to greyscale, edge detection, morphological transforms, circle detection), the working copy of the image is written to the working directory so that the user can see the intermediate steps in order to aid troubleshooting.

Results

Image	# Coins	TP	FP	FN	F1	Notes
1	13	11	4	2	0.786	Long run time (180s)
2	19	19	4	0	0.904	
3	9	9	2	0	0.9	
4	10	7	2	3	0.737	
5	12	12	1	0	0.96	
6	9	5	0	4	0.714	
7	10	10	1	0	0.952	
8	9	6	0	3	0.8	
9	10	8	0	2	0.889	Gold/silver coin; inner circle detected
10	12	12	0	0	1.0	
Total TP	%	85.7%				

Table 3: CoinDetector0 algorithm performance. Centers, radii, and detections are recorded in [image_root_name]report.txt files in the dataset.

Run Time

Run time for a single image can be several minutes of clock time, I observed clock times up to 4 min on a 16 GB/1.3-GHz/4 core Windows 10 machine in the supplied data set.

Appendix I: Installation

Visual Studio 2019

`CoinDetector0` was built for an x64 processor using a Visual Studio 2019 project named HelloOpenCV. Paths to the OpenCV include directories and libraries are set in the .sln and .vcxproj files. The executable file has been built in the x64/debug subfolder of the project directory.

cmake

a cmake file was generated from the .sln and .vcxproj files using cmake-converter. By: putting CoinDetector0.cpp in a directory, using the `cmake .` and `cmake –build` steps, the user can build CoinDetector0 from source.

Necessary edits to CMakeLists.txt

Set PATH variables `LOCAL_OPENCV_INCLUDE_DIR` and `LOCAL_OPENCV_LIB_DIR` to your system's paths to the OpenCV header files and libraries

```
125 set(LOCAL_OPENCV_INCLUDE_DIR "C:/Users/cl140/Downloads/opencv/build/
126 set(LOCAL_OPENCV_LIB_DIR "C:/Users/cl140/Downloads/opencv/build/x64/
127 LF
```

Other aspects of CMakeLists.txt involve the build platform and will need to be edited if you do not run Windows x64, i.e.:

```
147 ##############################################################
148 if("${CMAKE_VS_PLATFORM_NAME}" STREQUAL "x64")LF
149     set_target_properties(${PROJECT_NAME} PROPERTIESLF
150         TARGET_NAME_DEBUG   "CoinDetector0"LF
151     )LF
152 endif()LF
```

Repository

Source code for the solution is hosted on my GitHub: amatrhr/HelloOpenCV (github.com).

Appendix II: Images

Figure 1: Coin image thumbnails

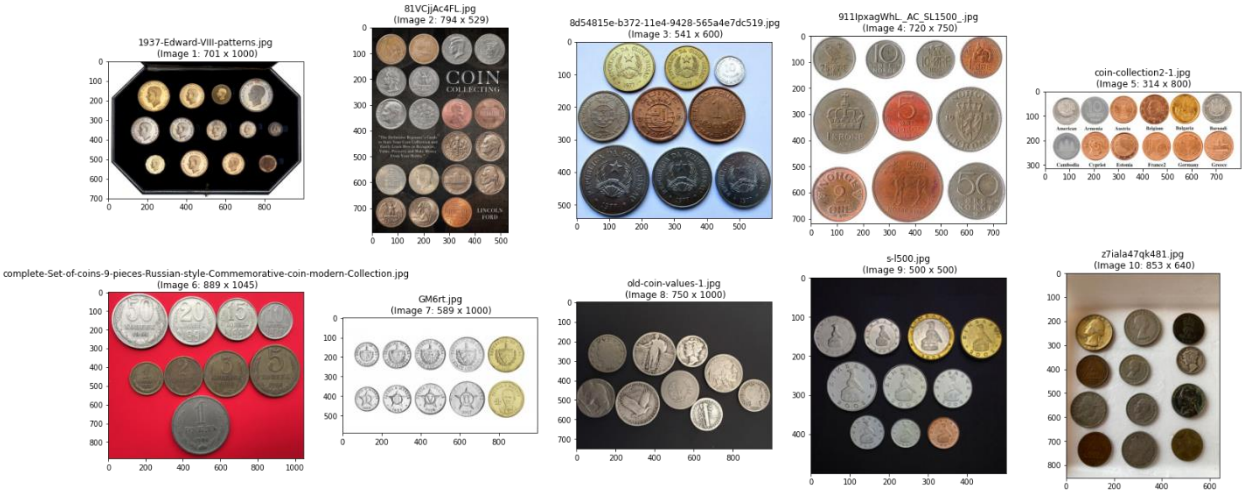

Figure 2: Coin Image Histograms

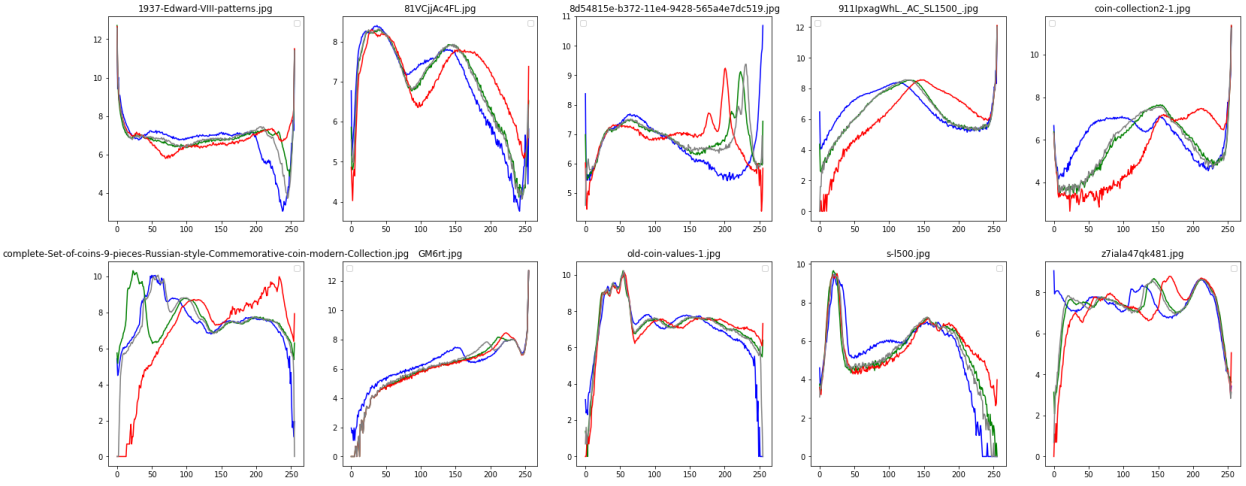

Figure 3: Detected Coins

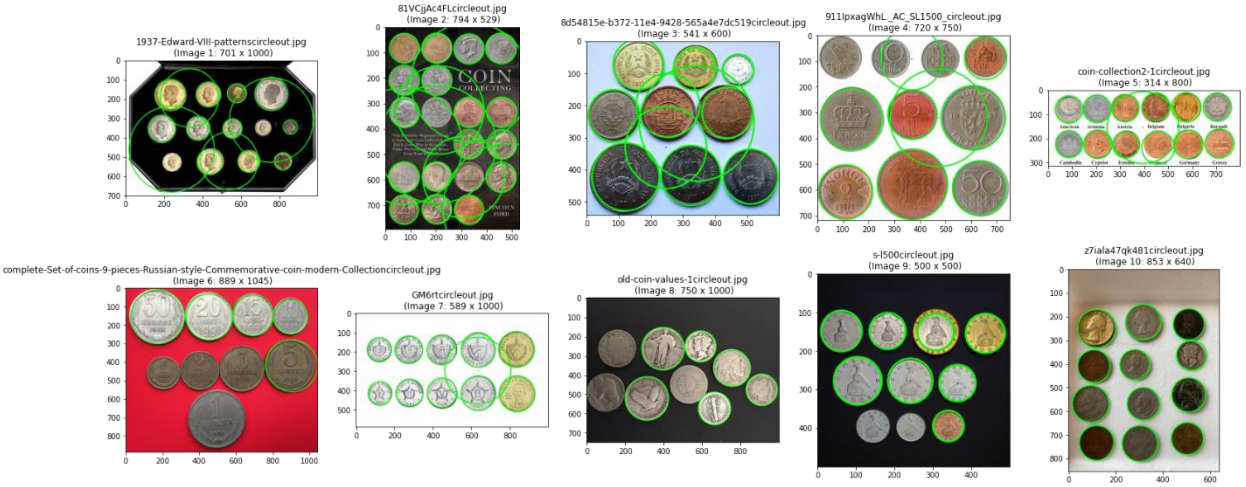

Figure 4: Example of Annotation Interface