Coin-Detection-Assignment

August 4, 2019

1 Coin Detection

So far we have studied about various mophological operations and different thresholding techniques in some detail. Now it's time to apply these concepts for a practical application - **Coin Detection**.

1.1 Aim

In this assignment, you will work with 2 different images (so 2 different parts) and will use **only** morphological operations and thresholding techniques to detect the total number of coins present in the image. Your submission will be graded based on your use of the concepts covered in this module, experimentation performed to achieve at your final solution, documentation, and finally, the total number of coins successfully detected in the images. Each part will be of 15 marks. This assignment will be entirely **manually graded** so make sure that you do NOT remove any experimentation you have done as well as the observation you made after each step.

Proper documentation for each step should be provided with help of markdown

1.2 Outline

The main steps that you can follow to solve this assignment are:

1. Read the image.
2. Convert it to grayscale and split the image into the 3 (Red, Green and Blue) channels. Decide which of the above 4 images you want to use in further steps and provide reason for the same.
3. Use thresholding and/or morphological operations to arrive at a final binary image.
4. Use **simple blob detector** to count the number of coins present in the image.
5. Use **contour detection** to count the number of coins present in the image.
6. Use **CCA** to count the number of coins present in the image.

We have also provided the results we obtained at the intermediate steps for your reference.

2 Assignment Part - A

2.1 Step 0: Include Libraries

```
In [1]: #include <iostream>
```

In [2]:
```
#include "../resource/lib/public/includeLibraries.h"
```

In [3]:
```
#include <opencv2/opencv.hpp>
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
```

In [4]:
```
#include "../resource/lib/public/matplotlibcpp.h"
#include "../resource/lib/public/displayImages.h"
```

In [5]:
```
using namespace std;
```

In [6]:
```
using namespace cv;
```

In [7]:
```
using namespace matplotlibcpp;
```

2.2 Step 1: Read Image

In [8]:
```
// Image path
string imagePath = DATA_PATH + "images/CoinsA.png";
// Read image
// Store it in the variable image
///
/// YOUR CODE HERE
///

Mat image = imread(imagePath, IMREAD_COLOR);

Mat imageCopy = image.clone();
```

In [9]:
```
plt::figure();
plt::imshow(image);
auto pltImg = displayImage(image);
```

In [10]:
```
pltImg
```

Out[10]:

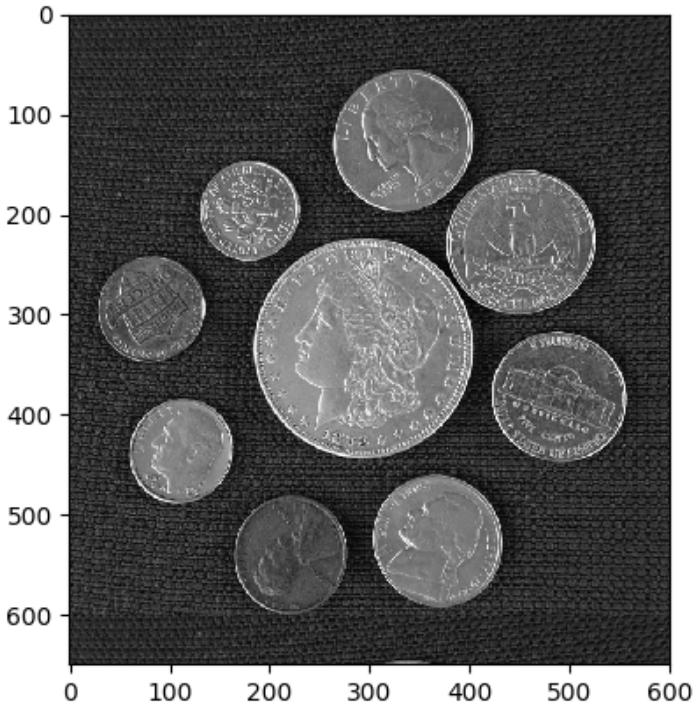
2.3 Step 2.1: Convert Image to Grayscale

In [11]:
```
// Convert image to grayscale
// Store it in the variable imageGray
///
/// YOUR CODE HERE
///

Mat imageGray;
cvtColor(image, imageGray, COLOR_BGR2GRAY);
```

In [12]:
```
plt::figure();
plt::imshow(imageGray);
pltImg = displayImage(imageGray);
pltImg
```

Out[12]:



2.4 Step 2.2: Split Image into R,G,B Channels

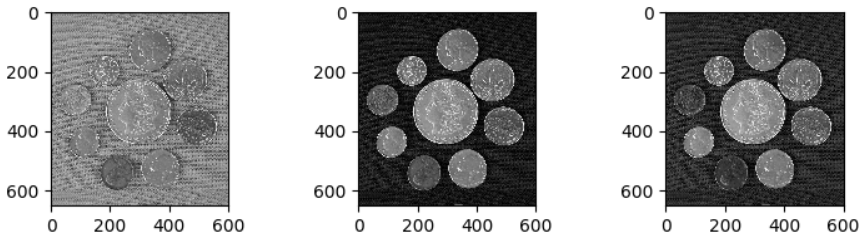
In [13]:
```
// Split cell into channels
// Store them in variables imageB, imageG, imageR
///
/// YOUR CODE HERE
///

Mat channels[3];
split(image, channels);
Mat imageB = channels[0];
Mat imageG = channels[1];
Mat imageR = channels[2];
```

In [14]:
```
plt::figure_size(900,200);
plt::subplot(1,3,1);
plt::imshow(imageB);
pltImg = displayImage(imageB);
plt::subplot(1,3,2);
plt::imshow(imageG);
pltImg = displayImage(imageG);
```

```
plt::subplot(1,3,3);
plt::imshow(imageR);
pltImg = displayImage(imageR)
```

Out[14]:



2.5 Step 3.1: Perform Thresholding

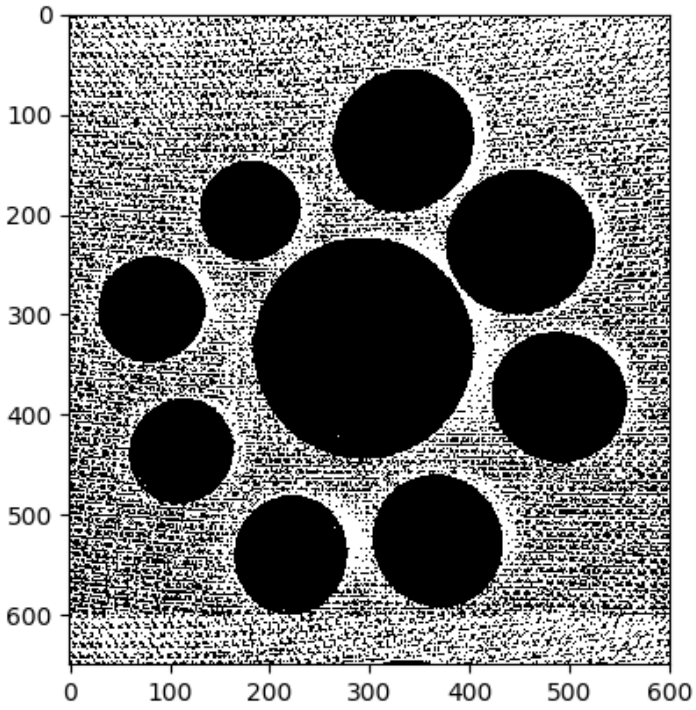
You will have to carry out this step with different threshold values to see which one suits you the most. Do not remove those intermediate images and make sure to document your findings.

In [15]:
```
///
/// YOUR CODE HERE
///
Mat src = imageG;
int thresh = 30;
int maxValue = 255;

Mat dst;
threshold(src, dst, thresh, maxValue, THRESH_BINARY_INV);
```

In [16]:
```
// Display image using matplotlibcpp
// We have provided the sample code here
// Please modify it as required
plt::figure();
plt::imshow(dst);
pltImg = displayImage(dst);
pltImg
```

Out[16]:



2.6 Step 3.2: Perform morphological operations

You will have to carry out this step with different kernel size, kernel shape and morphological operations to see which one (or more) suits you the most. Do not remove those intermediate images and make sure to document your findings.

In [17]:
```
///
/// YOUR CODE HERE
///
int kSizeDilate = 7;
Mat kernelDilate = getStructuringElement(MORPH_ELLIPSE, Size(kSizeDilate, kSizeDilate)
```

In [18]:
```
///
/// YOUR CODE HERE
///
Mat imageDilated;
dilate(dst, imageDilated, kernelDilate);
```

In [19]:
```
// Display image using matplotlibcpp
// We have provided the sample code here
// Please modify it as required
```