Abstraction (short) & Introduction

The proposed algorithm for coin counting have 4 different stages, which are pre-processing, coin segmentation, background removal and identification of gold and silver coin, and coin classification. Each stages will produce an output, for further processing.

Pre-processing stage involves creating the mask of the coin, which is a binary image where white represent the coin, and black represent the background.

The mask will then be passed to coin segmentation stage, which uses simple blob detector and watershed from opencv's library. This stage will then return a list of segmented images which contains only the coins.

The list of coins will then be passed to background removal, which will remove the corner or any background which may disrupt the accuracy later. After the background removal, the segmented coins will be further subset into gold and silver list.

The gold and silver list will then be passed to coin classification stage, which will classify the value of the coins and return the count and value of type of coin.

Description of methods

1. Pre-processing (binary image)
2. Coin segmentation (simple blob detector, watershed)
3. Background removal and identification of gold and silver coin (HSV color model)
4. Coin classification (template matching)

Description of Methods

1. Pre-processing

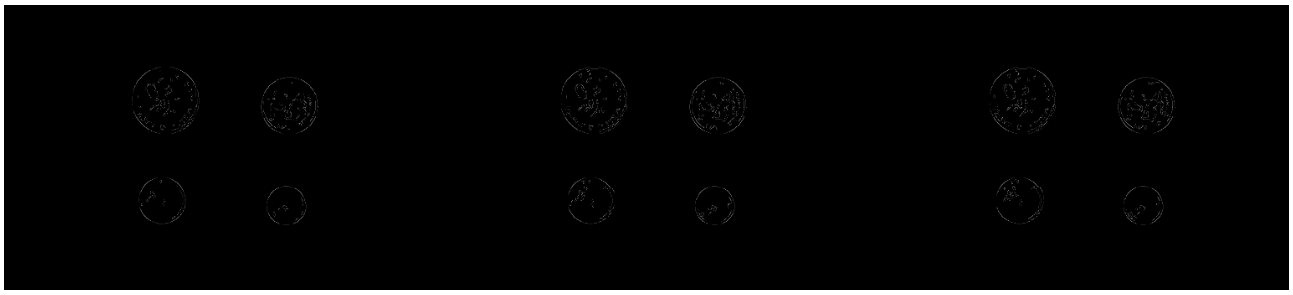


Illustration 1: Result of blue, green, red channel coin edges by applying Canny with upper bound threshold of 220, and lower bound threshold of 80.

Image is split into R, G, B channels, canny edge is then used on R, G, B channels to obtain binary image of the coin edges.

Bitwise or operation is then used on R, G, B edge binary images to close up the small gap between the coins. However there may still be edges that have very small gaps in between them. Therefore, morphology closing with rectangle kernel of 3 by 3 is used to close up small gaps without expanding the coin edges.

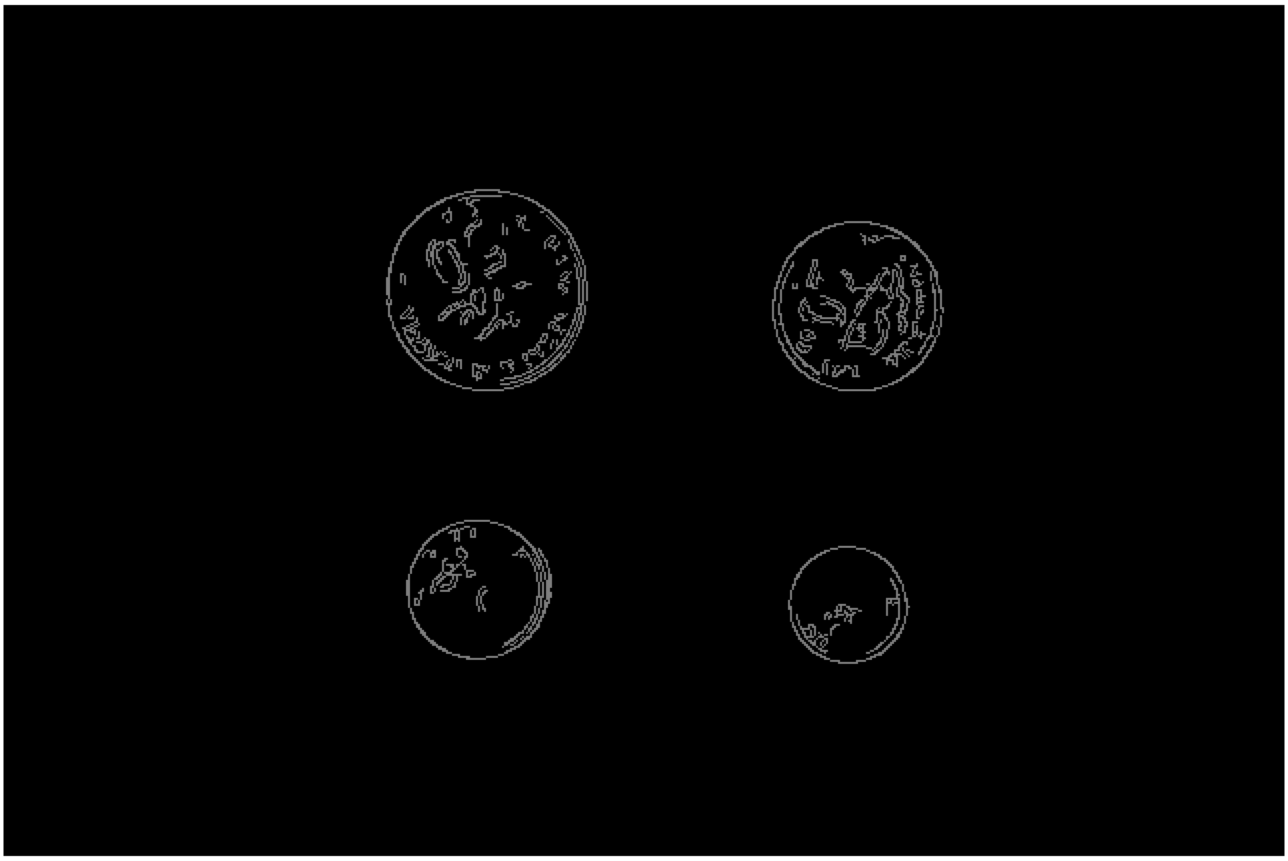


Illustration 2: Result of combining R, G, B edges.

Flood fill algorithm is used on most bottom left corner (0,0) with value of 255 to fill up all the background except for the coin. The output image of the flood fill is then inversed using bitwise not to highlight the coin's inside.

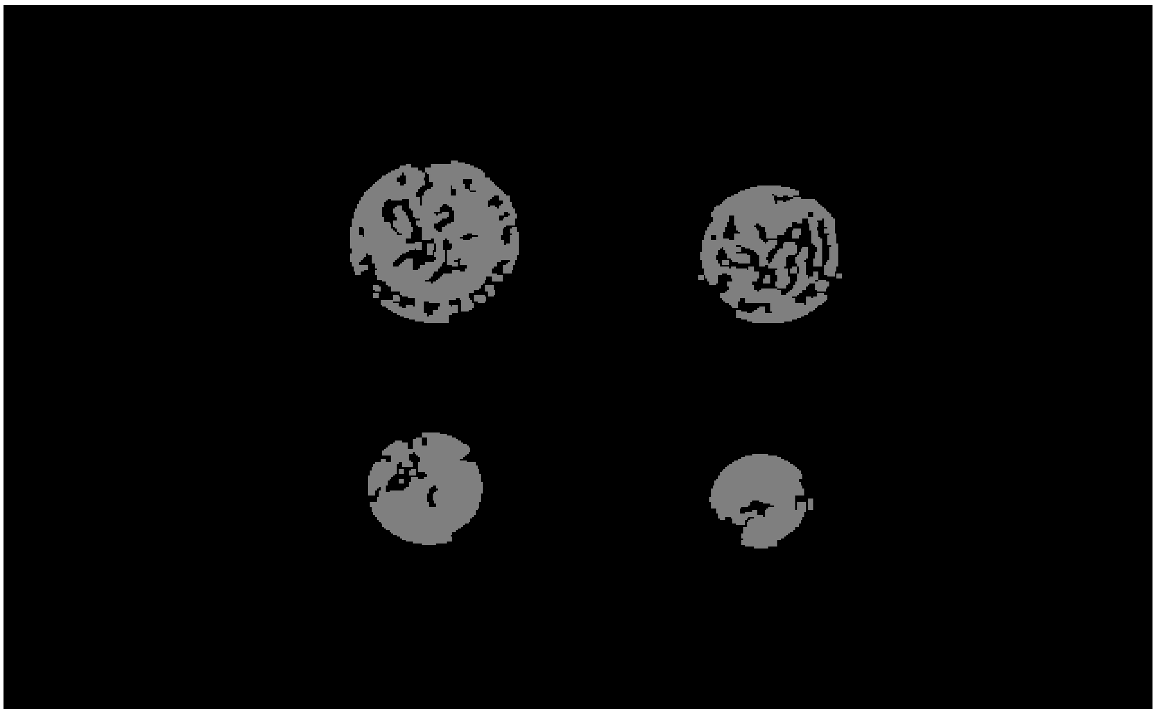


Illustration 3: Image shows flood fill is applied on the combined edges, and it is then inversed.

As shown on the image above, it only highlights the "body" of the coin but with no edges. Therefore, bitwise or is then used on the inversed flood filled with the combined canny's edge to give edges with body of coin.

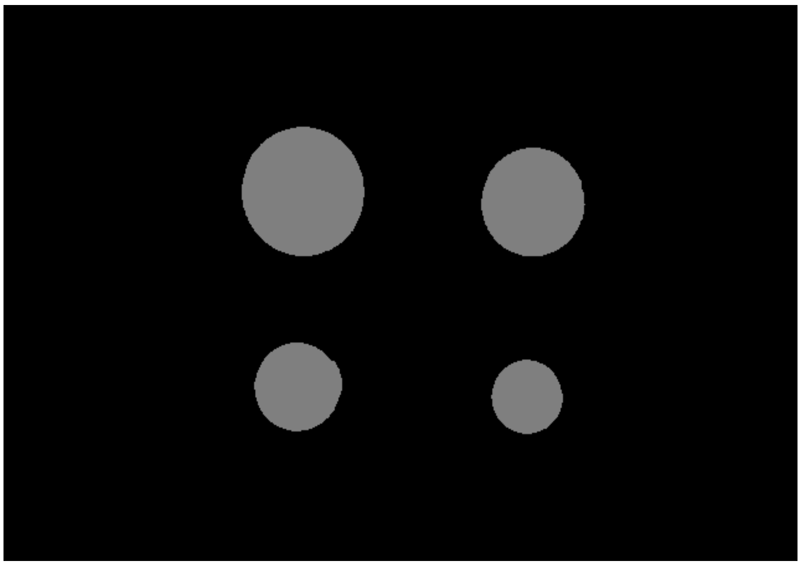


Illustration 4: Bitwise or of inversed flood fill and combine edges

Morphology erode with rectangle sized kernel of size 3x3 is then used to remove "small" and rough edges of canny and also noises.

2. Coin segmentation (simple blob detector, watershed)

Opencv2’s simple blob detector is then used on the coin’s mask to detect circles.

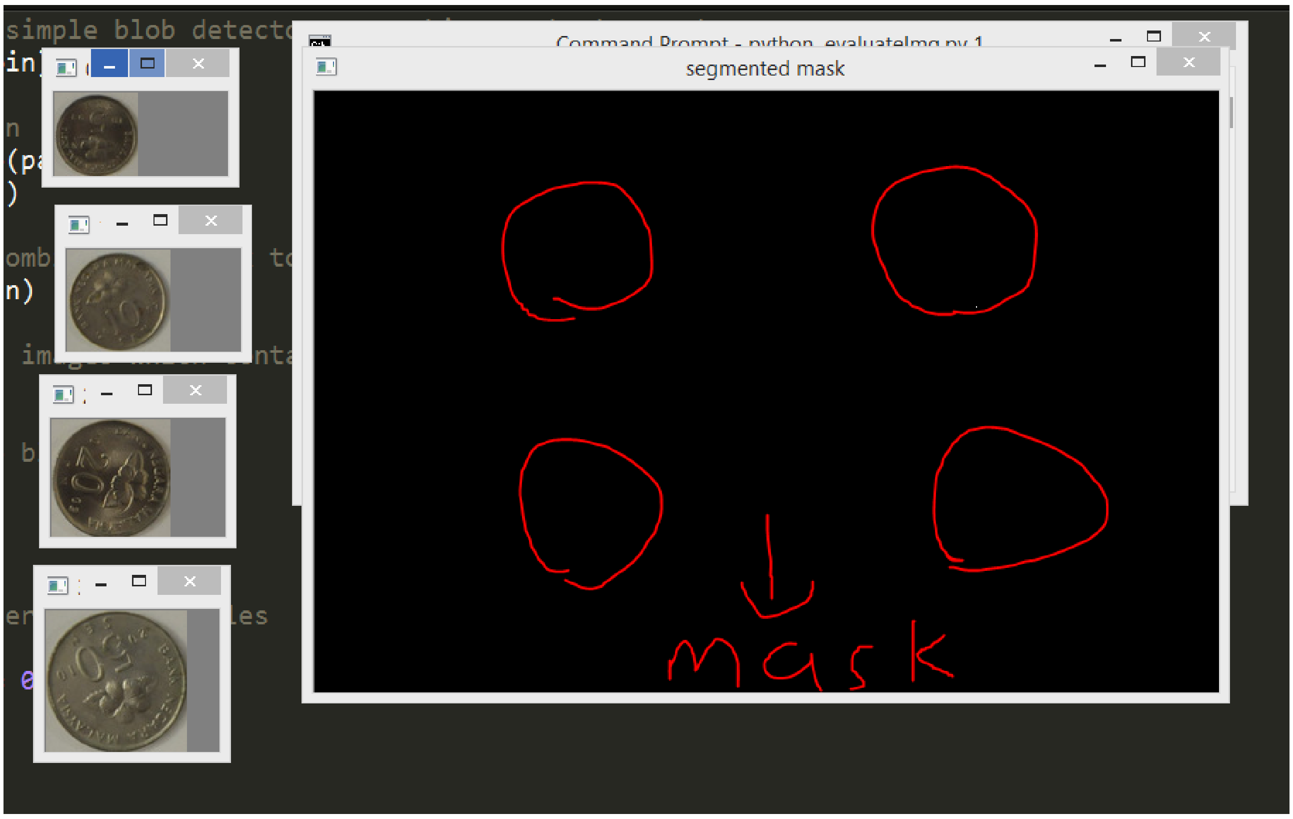
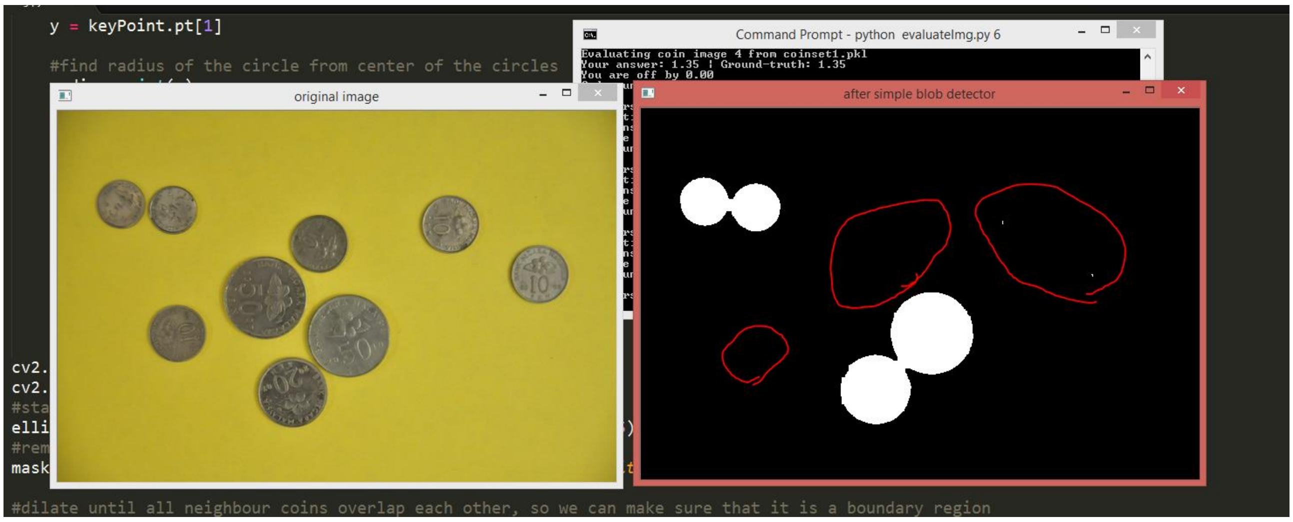


Illustration 5. Shows list of segmented coin and the mask after segmented.

Simple blob detector is applied to the mask (output of pre-processing step), to detect the circles. A list is then used to store all of the segmented circles obtained from simple blob detector, where for each of the “coin” that is segmented out, it is removed from the mask for watershed.


original image
after simple blob detector
Command Prompt - python evaluateImg.py 6

```
y = keyPoint.pt[1]

#find radius of the circle from center of the circles
```

```
Evaluating coin image 4 from coinset1.pkl
Your answer: 1.35 | Ground-truth: 1.35
You are off by 0.00
```

```
#dilate until all neighbour coins overlap each other, so we can make sure that it is a boundary region
```

Illustration 6. Image shows that coins that are closely connected to each other cannot be detected by simple blob detector, where the circled red represent the coins that are segmented out.

However, simple blob detector cannot detect coins that are connected. Therefore opencv2's watershed is used to segment out the coins that are connected.

Morphology opening of a disk structured kernel of 5x5 is used iteratively twice to remove small "island" noises. Dilation of 3 iterations is used to obtain the "sure background", and erosion of 8 iterations is used to obtain the center of coins "sure foreground" for watershed. Watershed will then return a 2d numpy array of labels, where each label represent one coin.

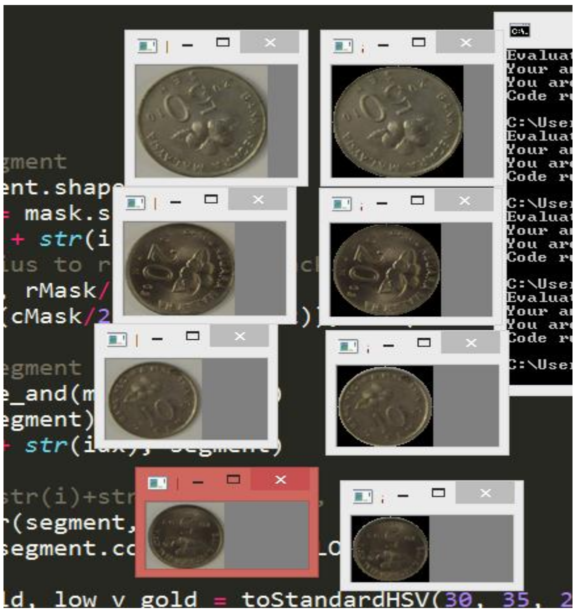
A dictionary is then used to store all coordinates of a type of label (labels = {key=[coord1, coord2, coord3, …]})

For each of the label (coin), the centroid is then calculated based on the formula below:
centroidX = maxLabelX – minLabelX
centroidY = maxLabelY – minLabelY

It is then added into the segmented coin list as specified above.

3. <u>Background removal and identification of gold and silver coin</u>

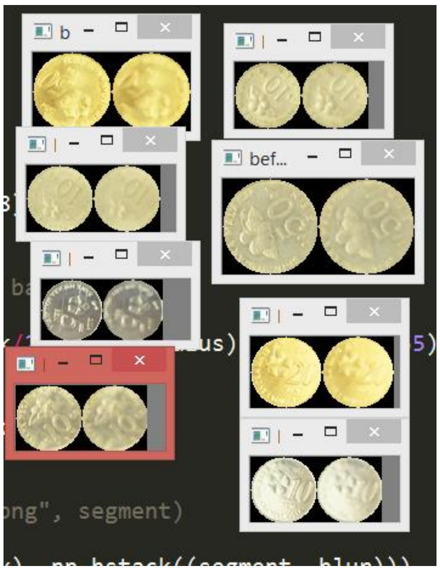
Background is removed by creating an white mask containing R(255), G(255), B(255), circular mask. Bitwise AND is then applied to each of the segmented coin list obtained from step 2, which will result in only the coin with no background.



```
gment
nt.shape
 mask.s
 + str(i
ius to r
 rMask/
(cMask/2
gment
_and(m
gment)
 str(i
tr(i)+str
(segment,
segment.cc
ld, low v gold = toStandardHSV(30, 35, 2
```

```
Evaluat
Your a
You ar
Code r

C:\User
Evaluat
Your a
You ar
Code r

C:\User
Evaluat
Your a
You ar
Code r

C:\User
Evaluat
Your a
You ar
Code r

C:\User
```

Illustration 7. Image shows that the left side are the coins with background, and the right with no background after the AND operation is applied.



```
us)
5)
ng", segment)
np.hstack((segment, blur)))
```

Illustration 8. Image shows before blur and after blur image. The right side's exposed light is slightly reduced.

Median blur of kernel size 3x3 is then used on the coin with no background to reduce down the exposed lights. The blurred segmented coin is then converted from BGR to HSV color model, a specified range of gold and silver is then defined whether if a coin is gold or silver. If the coin is gold, it will be added to a list of gold segment, otherwise it will be added to a list of silver segment.

4. **<u>Coin classification</u>**

Gold and silver coins are then concatenated side by side horizontally for template matching.

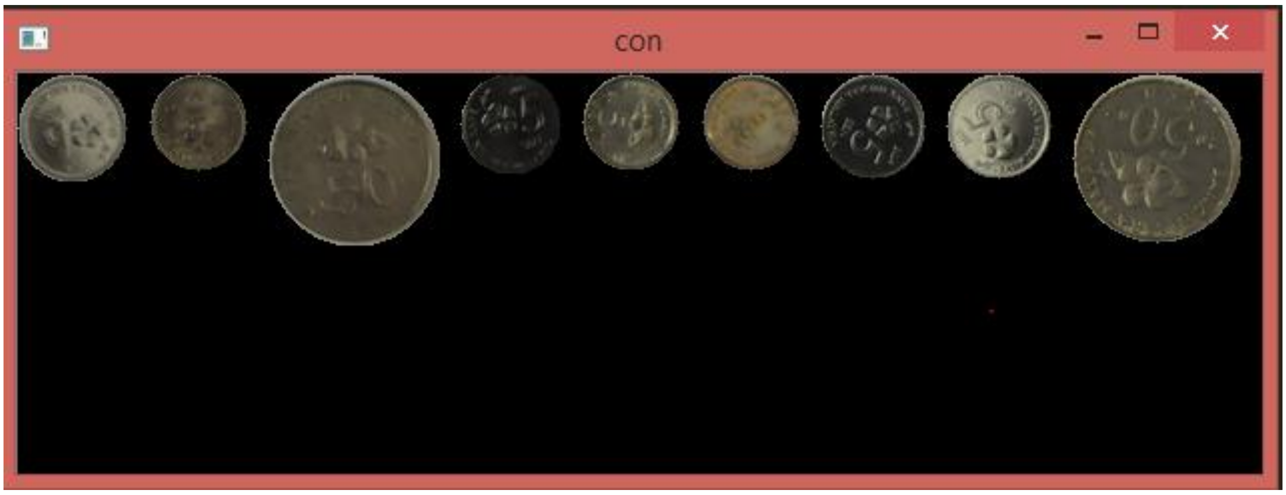
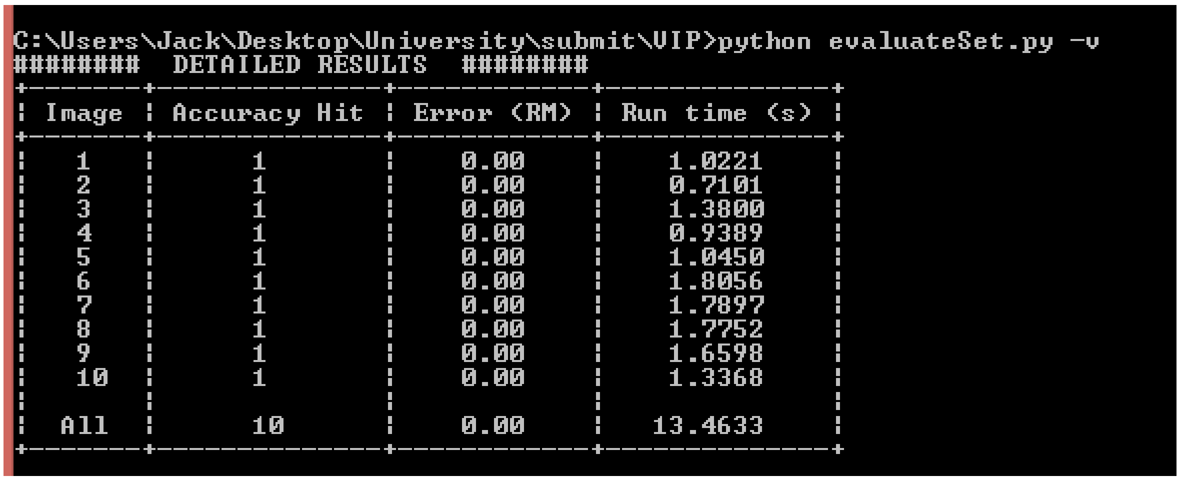


Illustration 9. Image shows that the coins are horizontally stacked side by side.

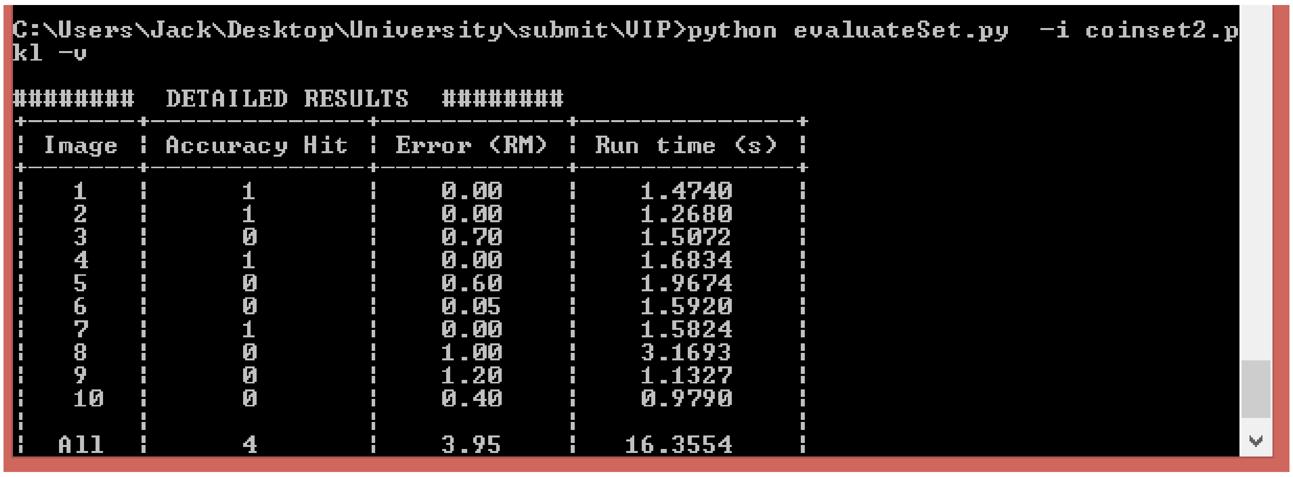
Template matching is then used to identify how much the coins are worth.

Result & Analysis



```
C:\Users\Jack\Desktop\University\submit\VIP>python evaluateSet.py -v
########  DETAILED RESULTS  ########
+-------+--------------+------------+--------------+
| Image | Accuracy Hit | Error (RM) | Run time (s) |
+-------+--------------+------------+--------------+
|   1   |      1       |    0.00    |    1.0221    |
|   2   |      1       |    0.00    |    0.7101    |
|   3   |      1       |    0.00    |    1.3800    |
|   4   |      1       |    0.00    |    0.9389    |
|   5   |      1       |    0.00    |    1.0450    |
|   6   |      1       |    0.00    |    1.8056    |
|   7   |      1       |    0.00    |    1.7897    |
|   8   |      1       |    0.00    |    1.7752    |
|   9   |      1       |    0.00    |    1.6598    |
|   10  |      1       |    0.00    |    1.3368    |
|       |              |            |              |
|  All  |      10      |    0.00    |   13.4633    |
+-------+--------------+------------+--------------+
```

Illustration 10. Image shows the accuracy of dataset 1.



```
C:\Users\Jack\Desktop\University\submit\VIP>python evaluateSet.py  -i coinset2.p
kl -v

########  DETAILED RESULTS  ########
+-------+--------------+------------+--------------+
| Image | Accuracy Hit | Error (RM) | Run time (s) |
+-------+--------------+------------+--------------+
|   1   |      1       |    0.00    |    1.4740    |
|   2   |      1       |    0.00    |    1.2680    |
|   3   |      0       |    0.70    |    1.5072    |
|   4   |      1       |    0.00    |    1.6834    |
|   5   |      0       |    0.60    |    1.9674    |
|   6   |      0       |    0.05    |    1.5920    |
|   7   |      1       |    0.00    |    1.5824    |
|   8   |      0       |    1.00    |    3.1693    |
|   9   |      0       |    1.20    |    1.1327    |
|   10  |      0       |    0.40    |    0.9790    |
|       |              |            |              |
|  All  |      4       |    3.95    |   16.3554    |
```

Illustration 11. Image shows the accuracy of dataset 2.

The accuracy was 100% for dataset 1, however it’s only 40% for dataset 2 and the total error for dataset 2 was only $3.95. The errors are mainly caused due to bad segmentation, which will be discussed later on. The template matching and coin color (gold and silver) classification still have close to 100% accuracy in dataset 2.

As bad segmentation is the main cause of the problem for error we’ll focus mainly on bad segmentations of coins.

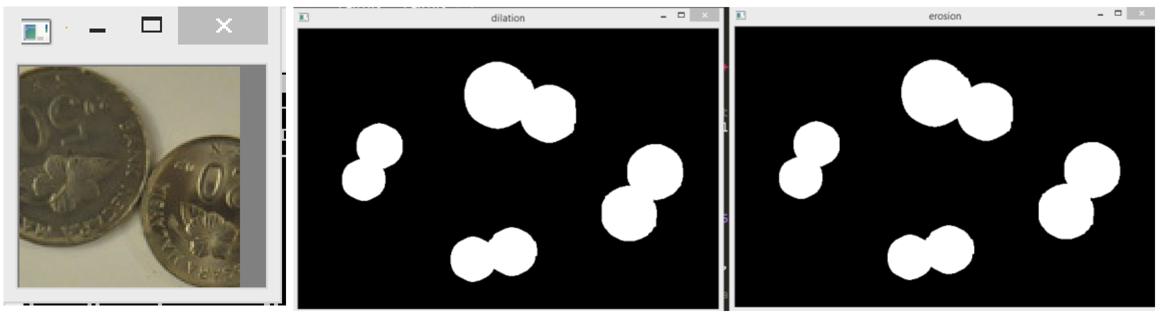


Illustration 12. Bad segmentation from watershed.

As seen above, the error is mainly from bad segmentation. The sure foreground still overlaps each other even after the erosion, which is then applied to watershed. The dilation and erosion does change have too much effect on the mask.

Another possible case maybe the erosion may erode coins with small radius such as 5 sen, which means watershed may not be able to detect the 5 sen, as it’s completely eroded by the morphology operation.

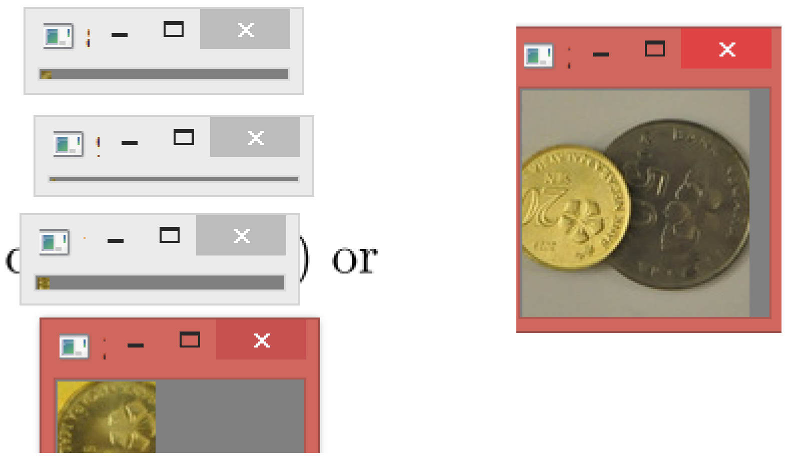


Illustration 13. Bad segmentation from simple blob detector.

Simple blob detector as seen above cannot detect coins that overlap each other, and also detected segments of coins that are not really circle.

Suggestions for Improvement

As most errors are resulted due to bad segmentation, improvements of the following may improve accuracy of segmentation:

- Add Hough circle transform to the coin segmentation process
- Fine tune parameters of simple blob detector
- Find a better and more improved version of morphology operation and different structure and size kernel

Furthermore, as the current dataset contains only small sets of coins. The performance of the program may decrease drastically due to template matching. Each template are matched with the stacked image, even though gold and silver coins are separated to improve the performance, the proposed algorithm's performance will suffer if the image contains too many coins.

Suggestion for classification of coins may improve the performance of algorithm:

- Filter bank may be used for pattern extraction
- Complex machine learning algorithm can be applied
- SIFT, SURF, ORB, Brute Force Matcher from opencv