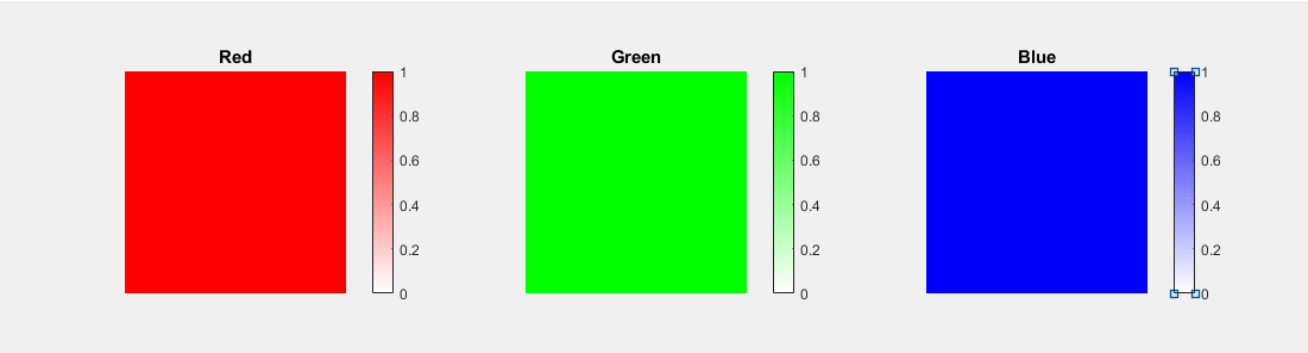
CS4302 Practical 3

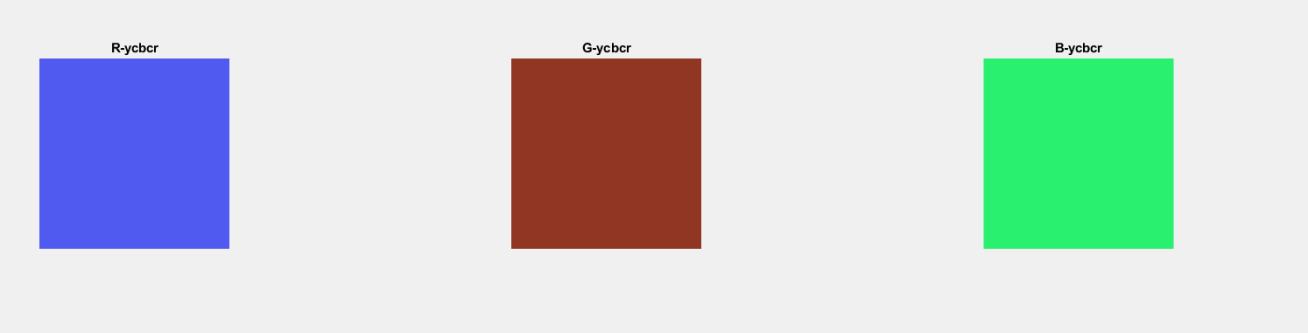
170025298 24/11/2020 Ruth Letham

Question 1



The leftmost figure shows the monochrome image filled with pure red. The RGB parameter of the colour is (255,0,0), where 255 here represents 1 in standard MATLAB form. The color bar represents the color from (0, 0, 0) to (255, 0, 0). Similarly, the second image is filled up with green colour (0, 255, 0) and the third image is filled up with blue (0, 0, 255). The parameter of these two figures in the MATLAB format represented as (0, 1, 0) and (0, 0, 1); and the colour bar represents the gradual change from pure white to the specific green or blue light channel.

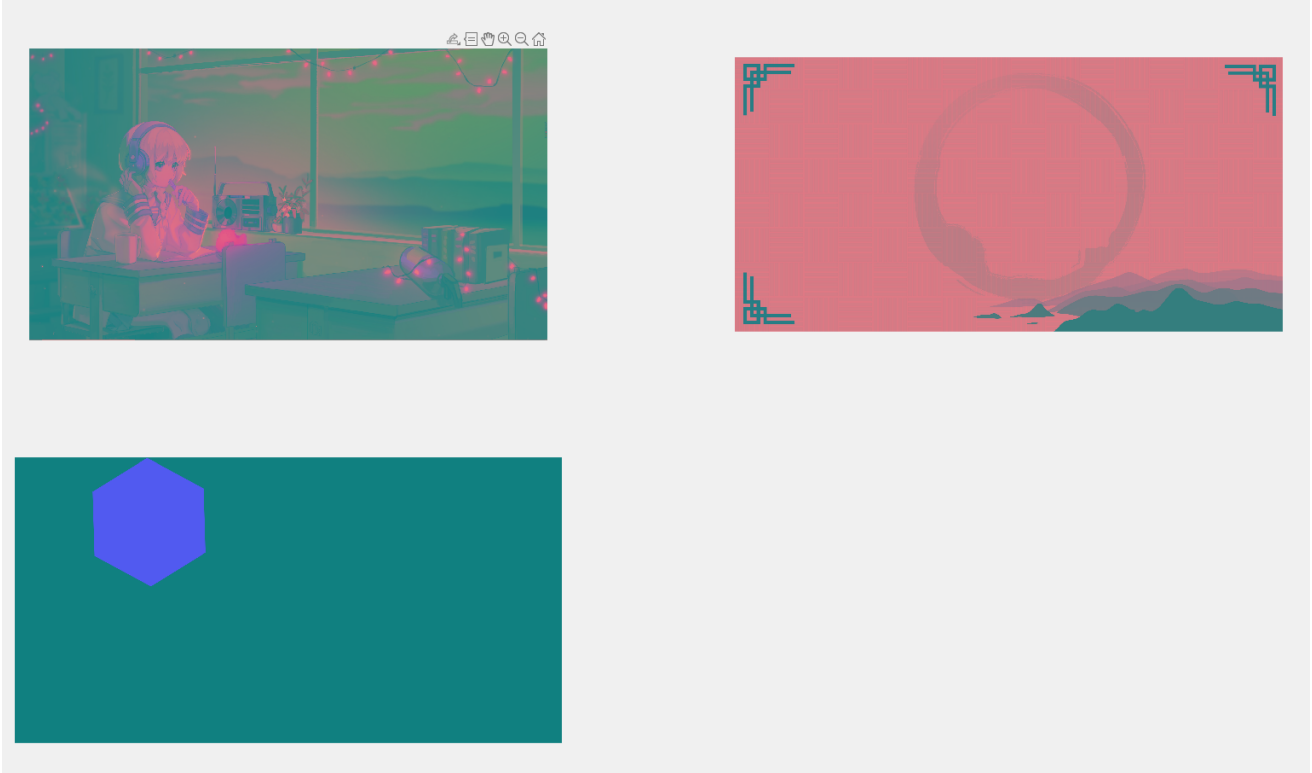
Question 2



The image that is showing above shows the result of Red, Green, Blue that are converted to YCbCr colour space. We can find out that the pure red is converted to blue, which has its property of the new RGB (81, 90, 240). This happened because the YCbCr image is constructed with "cat" Y, Cb, Cr. By the axiom of converting from RGB to YCbCr:

$$
\begin{aligned}
Y &= 0.299 \times R + 0.587 \times G + 0.114 \times B \\
Cb &= -0.172 \times R - 0.339 \times G + 0.511 \times B + 128 \\
Cr &= 0.511 \times R - 0.428 \times G - 0.083 \times B + 128;
\end{aligned}
$$

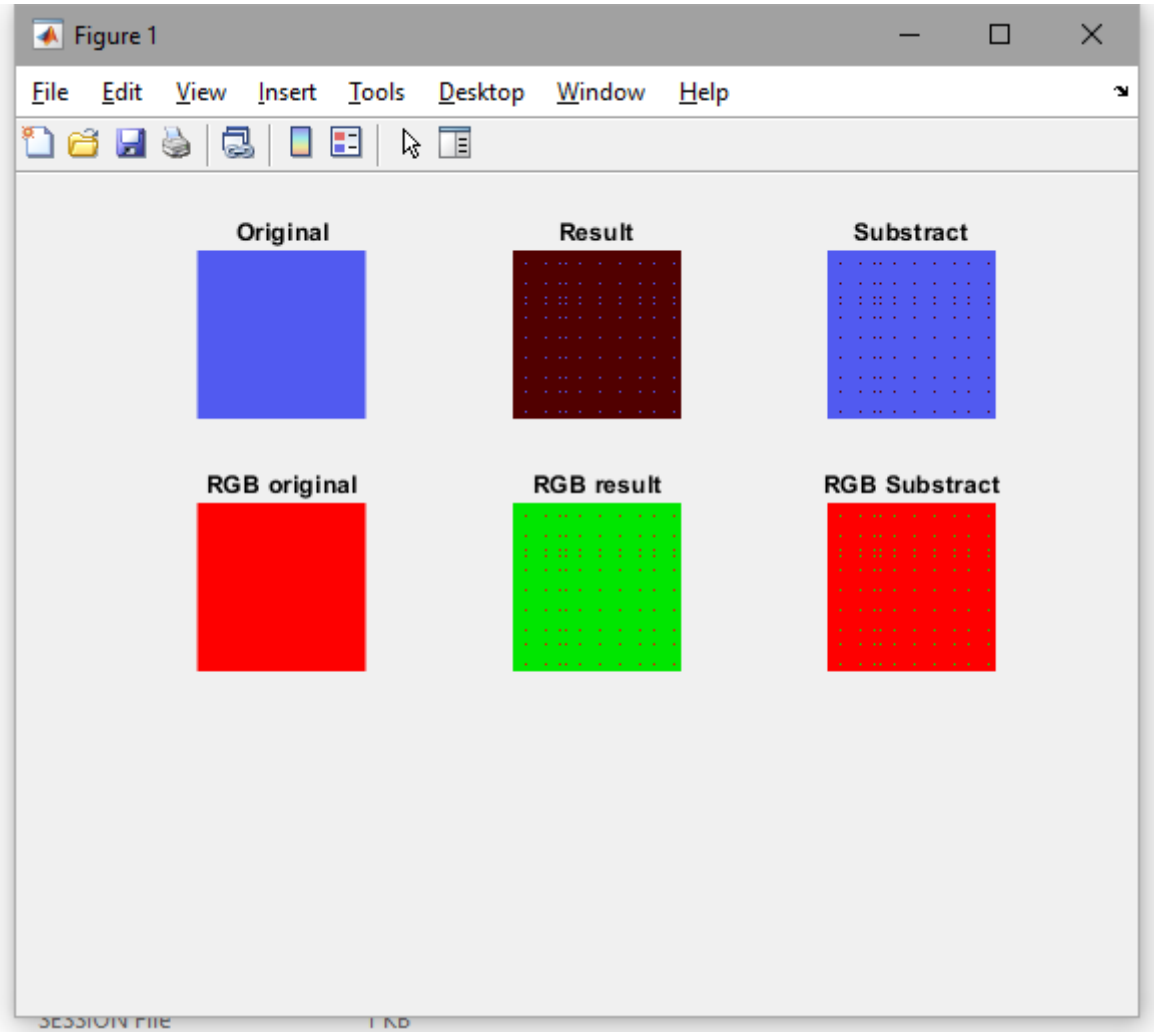
As the colour property is changed with axioms showing above, if we represent the image with YCbCr colour space in an RGB way, the colour of the image is undoubtedly changed.



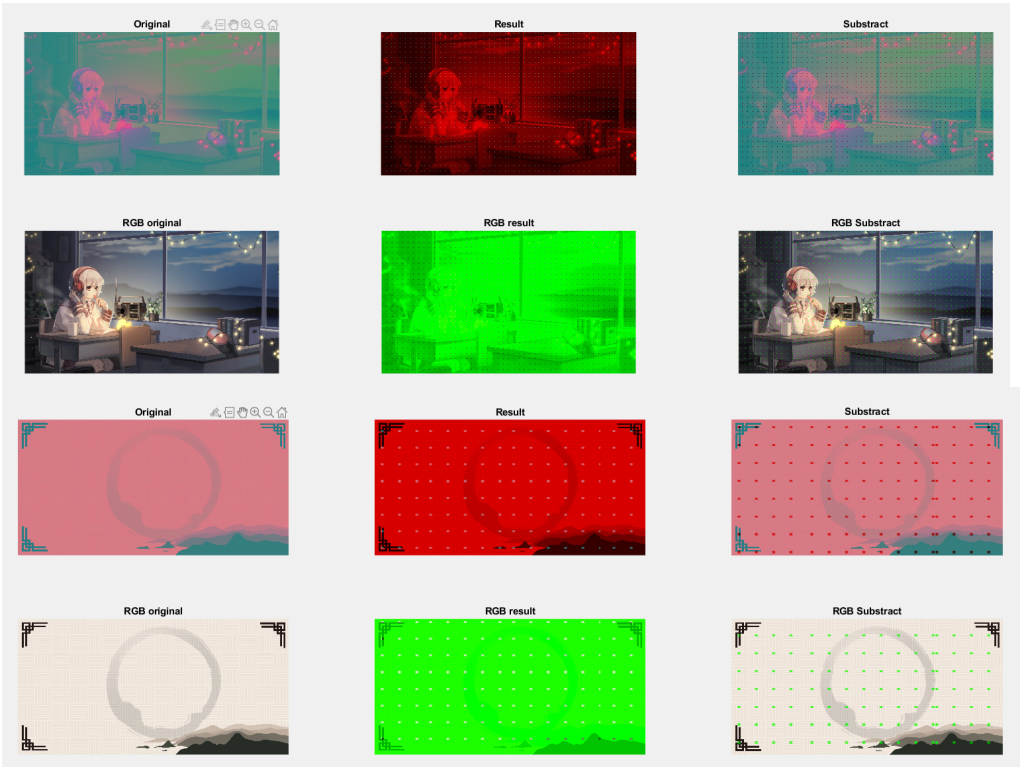
Similarly, the image showing above represents the result of converting images from RGB colour space to YCbCr colour space and showing in RGB mode. We can find out that the colour and colour depth is modified. However, the whole contour is not adjusted for two images showing top as the transformation of the image is proportionally modified. It applies to the single pixel, not to an entire image.

However, elements in the rasterized SVG image has changed if there is no filled element. This happens because the image contains a stroke of words, lines and rectangles and the colour information of the shape. During converting from RGB colour space to YCbCr colour space, the info of stroke is permanently destroyed. Moreover, there is no background colour for the image. Hence elements are lost in the SVG image.

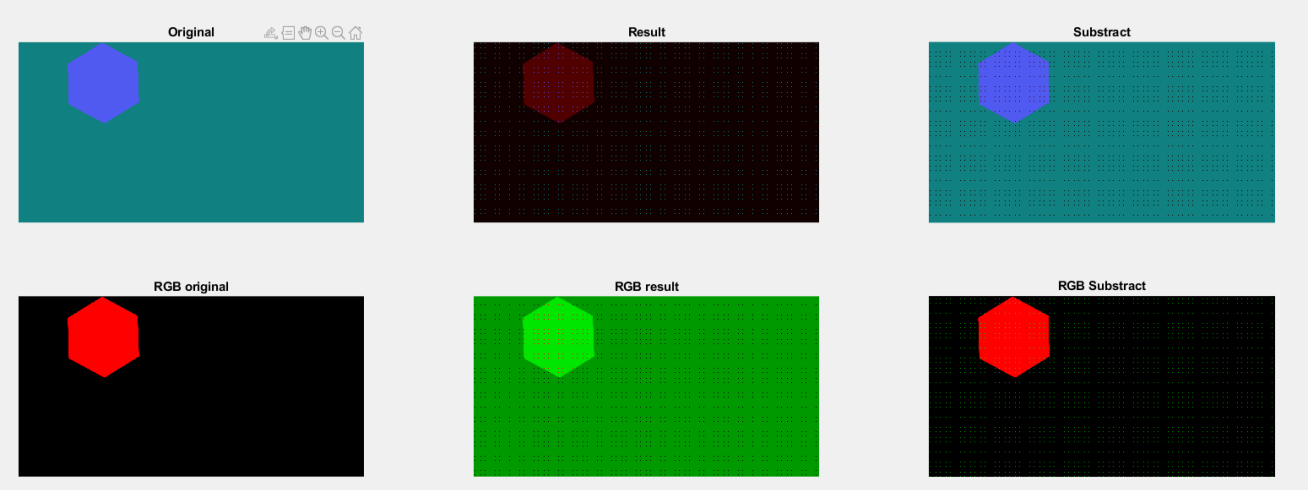
Question 3



The figure showing above shows the result of reducing the samplings of Cb and Cr by a factor of 8 of the original image and the subtraction between the original image and output image. Reducing sampling means that for every 8 pixels, one pixel is chosen as a sample, and other parts are filled with 0. We can find out that the subtraction between original and output image represents a new image with varieties of dots, where the colour of the dot is the colour of the original image and the background colour was changed. Similarly, by converting from YCbCr colour space into RGB, red is converted to green with varieties of the red dot.

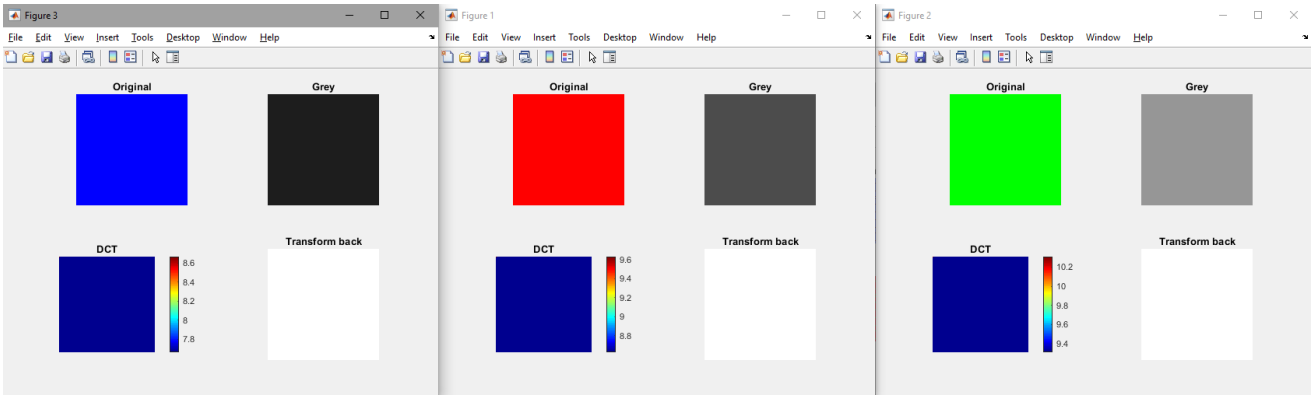


Similarly, the left figure shows the result of reducing sampling for a more complex image by a factor of 8, where the right figure shows the effect of lowering samplings for the less complicated image by a factor of 8. The conclusion is generally the same as operations done to the monochrome image.

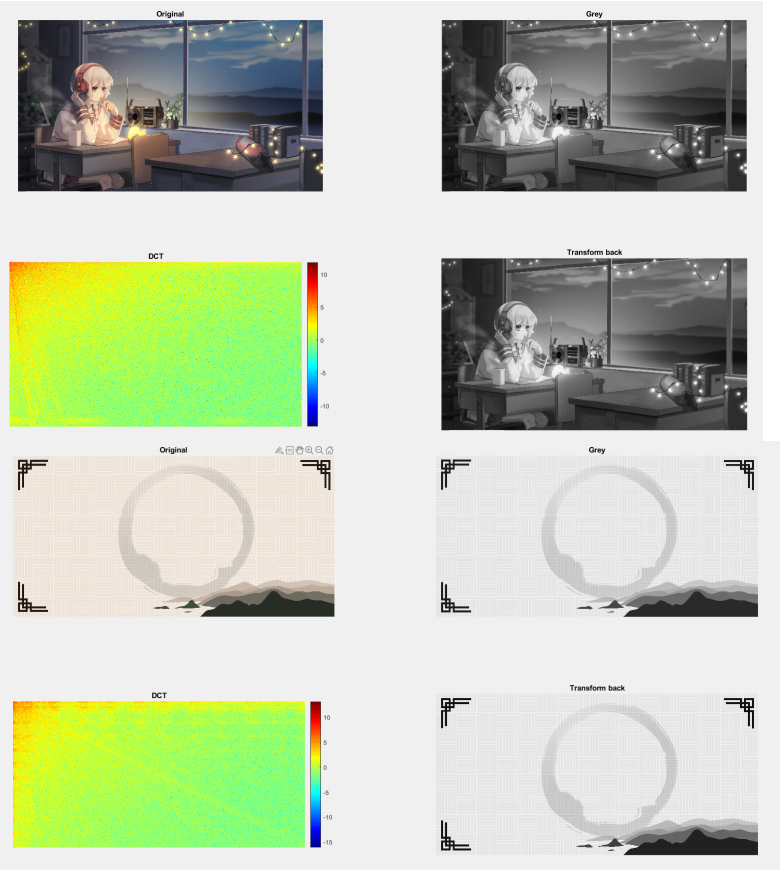


As the original image does not contain the background colour, words and rectangles are still invisible in the image showing above. However, the image should be well recovered.

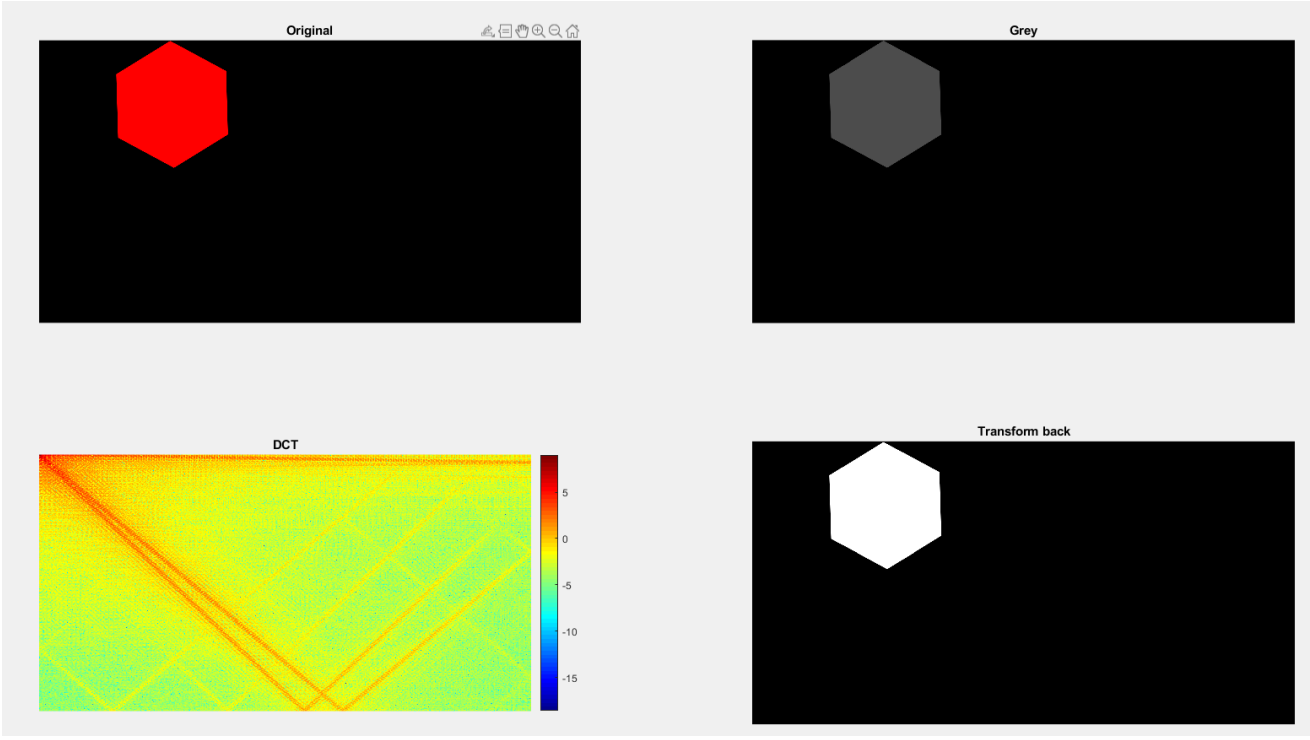
Question 4



The image showing above represents the original image, grey image, result of spatial frequency with DCT and transforming back image of monochrome images with pure Red, Green and Blue; as we can find out that the Grey image has different colour depth, whereas the blue has the darkest grey image. However, the frequency of these images is the same. The occurrence happens because the complexity of the image decides the frequency. As a result, DCT transforming leads to the same DCT result, which is 0. Hence the transforming back image is all white.



The UPPER image represents the information of the complex picture, where the LOWER image represents the information of the less complicated image. Since the spatial frequency domain means the complexity of the image, for an image with more variation, spatial frequency tends to be more complicated.



The spatial frequency for the SVG image is incredibly complex. Since the image is converted from vetorgram, details for the image are recorded carefully. Moreover, even words are hidden in the background; information does not lose. Hence the complexity of this image is incredibly high.

Question 5

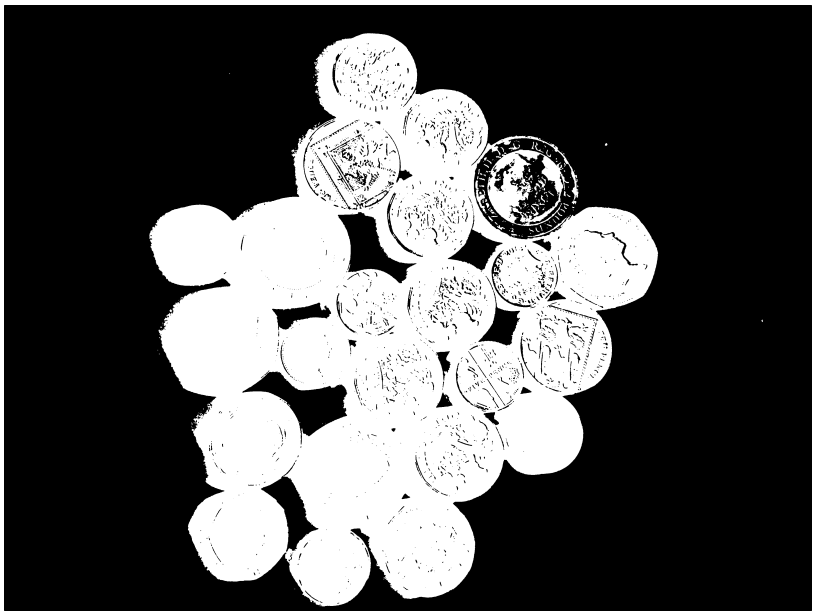
Methodology:

The simplest way of recognising coins without interruption is to recognise circles. If we can convert the shape of coins into just circles, counting could be easy.

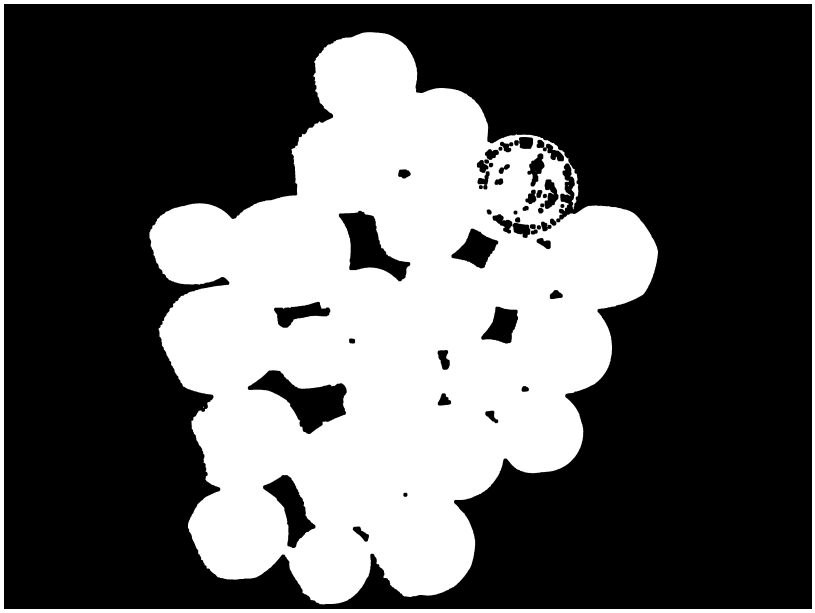
Image showing above is the original image which is represented in RGB colour space. Calculating and filtering images requires operate on grey colour space, hence, we would transfer the image into grey colour space.

We can find out that the background is a kind of interruption (top right, the light). As a result, reducing interruption is out next target. Since we would like to get only circles finally, we can decide a value of colour depth. If the colour in the image is more profound than this value, we can make the colour represents black; otherwise, the colour will represent white. The result is as following:

Due to the influence of light and the coin itself, there are still white spaces represents in the shot of the coin. To easily identify noises in the coin, we can make the coin represented in white and make the background represented in black. To get rid of noises represented above, we can choose a circle as a brush and wipe white spaces in the coin. We can also ignore the small black dot since the dot could never be the coin, just kind of noises. It is vital to decide the size of the eraser, as punctuation may cause and makes the coins connected.



Since there are still some sharp corners in the image. We would like to make the corner get mellow and full. By removing unnecessary part of the image, we can get the image as the following.



By calculating the white circle showing above, we can get the output for number of coins showing below:

```
>> Q5_1

N =

    23
```

For connected coins, we can define the minimum radius of the coin. Since the largest coin is no larger than the twice of the smallest coin, we can calculate the radius of the minimum coin and make the radius predefined. Finally we can calculate the number of coins.

Testing:

Easy:

As figure showing in methodology, the program can count the number of coins for this image correctly.

Medium:

```
count =

    24

N =

    18
```

As figure shows above, number of coins is 24, where the correct answer is 23. This happens because the program monitored radius from top to bottom and from left to right. We have not evaluated the number from top right to bottom left or other directions. As a result, one more coin is counted.

Hard and very hard:

```
count =

    14


N =

     9
```

```
count =

    16
```

Similarly, since the direction of counting is limited, the number of coins for hard is 14, and for very hard is 16. The program only count coins in the specific direction. As a result, the number is 14.

This program does not works for extreme, so there is no need to test the extreme here and in question 6, as the image cannot recognise the overlay coins,

Question 6

Methodology:

From question 5, we can evaluate radius for coins. Since coins in money_easy.jpg are all independent, we can evaluate the radius for each coins to evaluate which coins is for the coin.

```
len =

  Columns 1 through 7

   160   176   218   138   176   140   176
   181   213   228   159   212   157   206

  Columns 8 through 14

   204   181   135   214   135   159   134
   221   201   149   240   149   174   150

  Columns 15 through 21

   173   188   218   184   197   202   174
   201   205   240   201   220   220   194

  Columns 22 through 23

   158   174
   170   204
```

We make a predefined list that stores the size of the coin, and compare the size of coins to the predefined value.

Then, we would produce a map to identify the value of the coin. The map would be:

Value (pence)	Radius Min	Radius Max	Radius (.floor)
1	158	181	169
2	204	221	212
5	134	159	146
10	184	201	192
20	159	174	166
50	197	228	212
100	173	213	193
200	218	240	229

Since the radius for 2 pence and 50 pence are the same, we would regard these two coins as 26 pence in average when counting the value. By resorting coins from small to large, we would get the following:

```
pre_red = [146, 5; 166, 20; 169, 1; 192, 10; 193, 100; 212, 26; 229, 200];
```

And we can get the information about the smallest coin, which has its radius: 146. It is easy to decide which hole that fits the coin: we can subtract the radius of the hole to the radius of each pre-defined radius of coins. We can take the minimum non-zero term from the result to get the coin that is the most suitable to the hole and apply the coin to the hole. For holes that are connected, we can take measures that we have taken on counting holes, and subtract results for each step and check whether the rest of the hole that is larger than the radius of the smallest coin. After knowing about the number of coins and the value of coins, by multiplying the value of the coin and the number of coins, we can get the value of coins.

Testing:

money_easy:

```
pred =

     8     5
     0    20
     9     1
     0    10
     3   100
     3    26
     0   200


totalVal =

   427
```

The data showing above is the output of identifying. We can find out that the program failed to identify 200, 10, 20 pence. This happens as the radius of the hole is not well-distributed. The identifier does only recognises the first coin vertically; however, the shadow increases the radius of the coin, which prevented from the program identifies the larger coin. Hence the total value of coins is less than the exact amount of the coin.

money_medium:

As we can find out that the result is something that a bit more accurate when identifying the harder image. This happens because the hole represents larger and identifying can be easier.

money_hard (UPPER) and money_very_hard (LOWER):

```
pred =

     2     5
     0    20
     1     1
     1    10
     2   100
     1    26
     4   200


totalVal =

        1047
```

```
pred =

     0     5
     0    20
     1     1
     0    10
     0   100
     0    26
     8   200


totalVal =

        1601
```

Since counting is not entirely accurate when the program identifying the money, the result is a bit confused and can not be accurate either as showing above. The work is meaningless, and identifying coins here is entirely impossible.

money_extreme:

Since counting cannot identify overlay coins, the evaluation cannot identify overlay coins either: the coin identifying technique for two questions are the same.