Rafałowski Michał

Seminar on ICT

The main goal of the project is to develop a program, which allows hiding information in a PNG image. The algorithm was written in Python 3 language with a use of PIL module, which provides basic image operations. One symbol (8bits) is encrypted in one row of pixels, and therefore the amount of bytes of a binary file has to smaller than the length of an image. The starting point of encoding is in the right-bottom corner. Every symbol is hidden in 16 pixels.

The encoding algorithm (for RGB images) works as follows:

1. The sequence is opened as hex sequence, and then converted into binary sequence. The sequence is filled with a specific amount of zeroes if its length is smaller than 8

```
with open("test.png", "rb") as binary_file:
    b = binary_file.read()

msg = b2b(b)
```

2. The algorithm checks image mode, then choose specific encoder

```
if img.mode == "RGB":
    RGB_koder(img, msg)
```

3. An image is rotated by 180°

```
img = img.rotate(180)
```

4. Every 8-bit symbol is hidden in one row
5. Every even pixel in a row is a reference pixel. That pixel provides its parameters e.g. R, G and B parameters for RGB mode

```
# parameters of pixel[x,y]
r, g, b = img.getpixel((x, y))
```

6. Other pixel represents bits
7. If even pixel’s parameters are R,G,B, then:

   For bit = ‘1’ --- next pixel’s parameters are R+1, G+1, B+1

   For bit = ‘0’ --- next pixel’s parameters are R+2, G+2, B+2

```
if msg[licznik] == '1':
    img.putpixel((x+1, y), (r + 1, g + 1, b + 1))
elif msg[licznik] == '0':
    img.putpixel((x+1, y), (r + 2, g + 2, b + 2))
```

8. The encoding is ended with a pixel, that parameters are R+3, G+3, B+3

```
# leng - amount of bytes
img.putpixel((1, leng + 1), (r + 3, g + 3, b + 3))
```

9. Of course there are added exceptions. For instance what if R = 255? Then we have to subtract R value instead of adding.

```
elif r >= 253:
    if msg[licznik] == '1':
        img.putpixel((x+1, y), (r - 1, g + 1, b + 1))
    elif msg[licznik] == '0':
        img.putpixel((x+1, y), (r - 2, g + 2, b + 2))
```

10. An image is rotated again, and saved as "zakodowany.png"

```
img = img.rotate(180)
img.save("zakodowany.png")
```

11. After successful encoding, the message "Zakooczono kodowanie" appears on screen.

```
print("\n Zakończono kodowanie.")
```

The decoding algorithm (for RGB images) works as follows:

1. An image is rotated by 180°
2. The first loop's goal is to find message length by looking for a pixel with R+3,G+3,B+3 values (in best case). Now the algorithm knows how many rows it has to analyze

```
# wysokosc - length of image
for k in range(wysokosc):
    r, g, b = img.getpixel((0, k))
    r_k, g_k, b_k = img.getpixel((1, k))

    if (r_k == r-3 and g_k == g+3 and b_k == b+3) or \
            (r_k == r+3 and g_k == g-3 and b_k == b+3) or
\
            (r_k == r+3 and g_k == g+3 and b_k == b-3) or
\
            (r_k == r-3 and g_k == g-3 and b_k == b+3) or
\
            (r_k == r-3 and g_k == g+3 and b_k == b-3) or
\
            (r_k == r+3 and g_k == g-3 and b_k == b-3) or
\
            (r_k == r-3 and g_k == g-3 and b_k == b-3) or
\
            (r_k == r+3 and g_k == g+3 and b_k == b+3):

        d = k-1
        break
```

3. In the next loop, 2 neighbor pixels are compared, and then a specific value (‘0’ or ‘1’) is written into a temporary list

```
# off - an amount of reference pixels
if (r_x == r-1 and b_x == b+1 and g_x == g+1) or \
       (r_x == r+1 and b_x == b+1 and g_x == g-1) or \
       (r_x == r+1 and b_x == b-1 and g_x == g+1) or \
       (r_x == r - 1 and b_x == b + 1 and g_x == g - 1)
or \
       (r_x == r - 1 and b_x == b - 1 and g_x == g + 1)
or \
       (r_x == r + 1 and b_x == b - 1 and g_x == g - 1)
or \
       (r_x == r - 1 and b_x == b - 1 and g_x == g - 1)
or \
       (r_x == r+1 and b_x == b+1 and g_x == g+1):
    dec[x-off] = "1"
```

4. These temporary lists creates one final binary sequence

```
txt = txt + dec
```

5. Final binary sequence is convert into hex sequence

```
final = bitstring_to_bytes(final)
```

6. The hex sequence is written to new “result” file

```
f = open('result', 'wb')
f.write(final)
f.close()
```

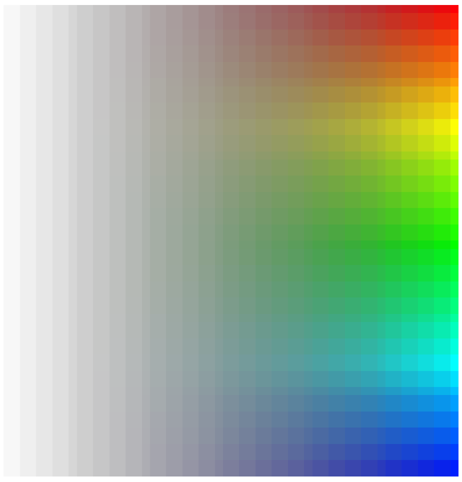
7. One of “Fleep” module’s functions gives a file’s extension based on the hex sequence, and then the name of “result” is changed (e.g. “result.png”)

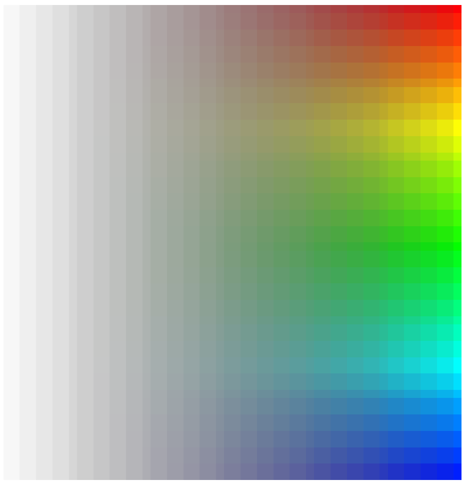
```
with open("result", "rb") as file:
    info = fleep.get(file.read(128))

f = str(info.extension[0])

thisFile = "result"
base = os.path.splitext(thisFile)[0]
os.rename(thisFile, base + "." + f)
```

8. An exception:
   If we want to encode a simple text message, it should be written in “.txt” file. “Fleep” module does not support “.txt” file extension, so the message is just printed on screen.

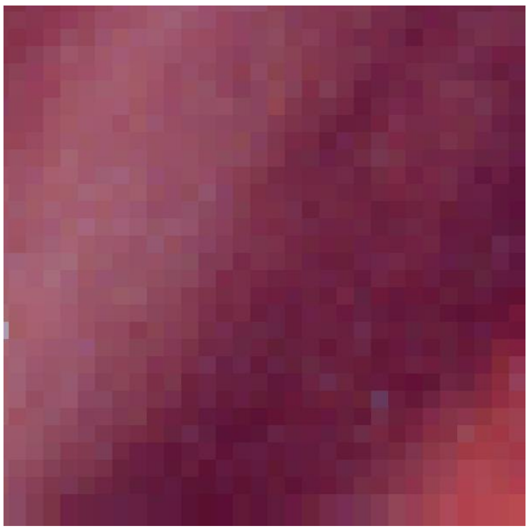
```
if len(info.extension) == 0:
    print("\n\n", str(final)[2:-1])
```

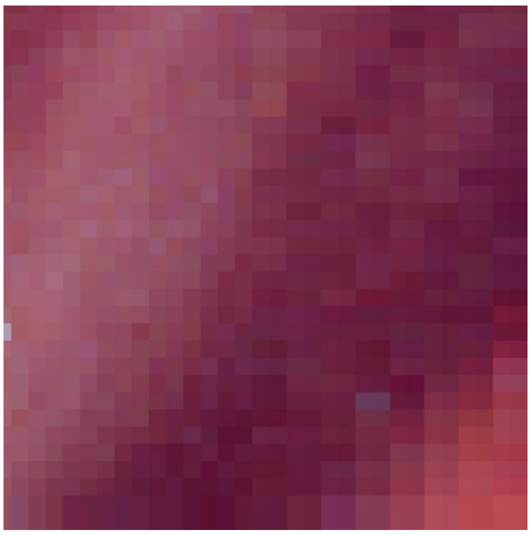
The Comparison:

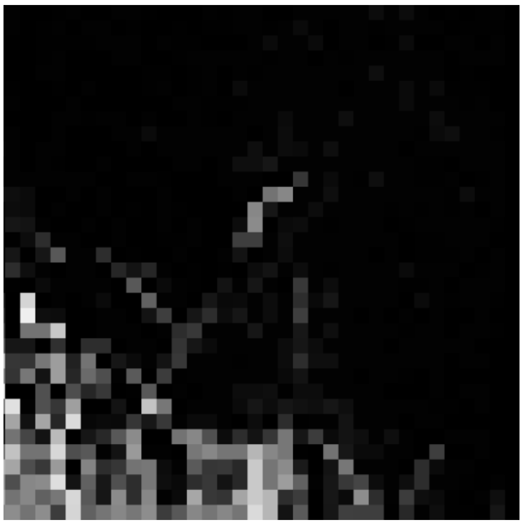


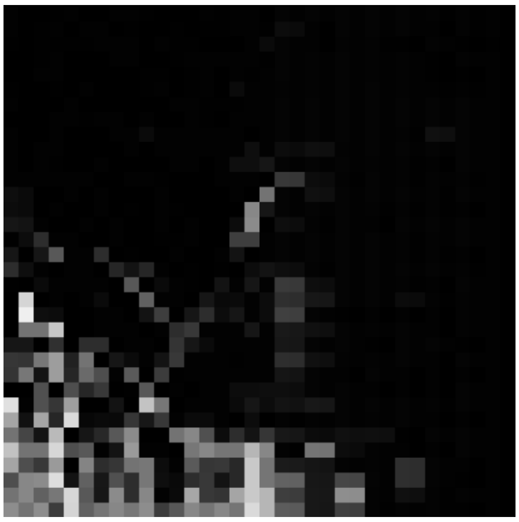


The coded binary file (in this case a PNG image) is decoded perfectly. The left one is a coded image, and the right one is decoded “result.png” image.









The left column represents initial images, while the right one represents encoded images. As it can be seen, our algorithm works better for darker images rather than bright ones, because 2 neighbor pixels' parameters contain almost the same color values and it makes the initial image's right side a bit blurry.

Summary:

Developing the code helped me understand the basics of steganography and the potential of it. It is an interesting field, but also hard to study, because little information is available. I am aware, that the method is not perfect, but it works and can be expended later on.