

Stéganographie sur le bit faible

Bernard léo | Oliveira da silVa nathan

NGUYEN VERGER | Image processing | 7/11/2022

Table des matières

[I. Introduction 1](#_Toc118370708)

[II. Le code 2](#_Toc118370709)

[1. Cacher l’image secrète 2](#_Toc118370710)

[2. Décoder l’image secrete 3](#_Toc118370711)

[III. Comparaisons 4](#_Toc118370712)

[IV. Stéganographie sur image en couleur 5](#_Toc118370713)

# Introduction

La stéganographie consiste à cacher une image secrète dans une autre image. Nous appellerons l’image secrète « image secrète » et l’autre image, « image cachette ».

Nous utiliserons les bits de poids faible afin de cacher l’image secrète. Nous remplacerons donc les bits de poids faible de notre image cachette les bits de poids fort de notre image secrète.   
La stéganographie est réalisée sur des images de niveau de gris et en couleur ainsi, qu’avec un nombre de bit variable pour pouvoir comparer les différences.

Pour réaliser ce projet, nous utiliserons Python 3.9.

# Le code

## Cacher l’image secrète

Pour commencer, nous supprimons les n derniers bits de l’image cachette et les remplacer par des 0 pour faire de la place pour ceux de l’image secrète.

Une image contenant texte

Description générée automatiquement

Ensuite, nous décalons les bits de poids fort de l’image secrète avec la même méthode que précédemment :

Une image contenant texte

Description générée automatiquement

Une fois cela fait, nous n’avons plus qu’à fusionner les deux images modifiées pour finir la stéganographie :

Une image contenant texte

Description générée automatiquement

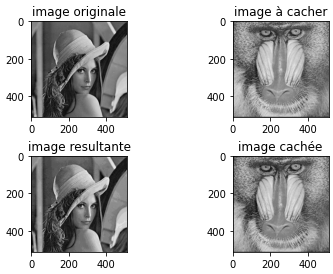
## Décoder l’image secrete

Pour décoder l’image, nous utiliser une autre fonction qui permet de récupérer les bits qui nous intéresse de l’image avec une opération « et ». Nous comparons les bits d’un pixel avec un nombre binaire de la forme 0000 1111 avec autant de 1 qu’il y a de bit à récupérer. Pour finir, nous redécalons les bits obtenus vers la gauche pour avoir une image normale.

Une image contenant texte

Description générée automatiquement

Cela donnera donc comme résultat :



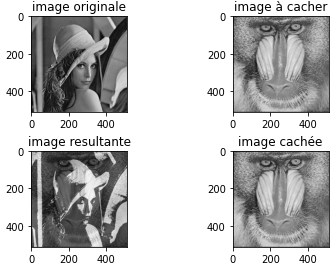
Pour appeler les différentes fonctions, nous utilisons la fonction map de python :

Une image contenant texte

Description générée automatiquement

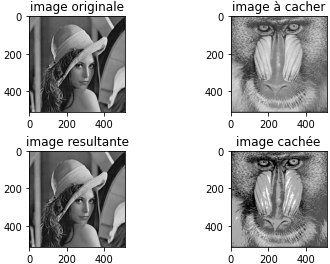
# Comparaisons

Pour montrer notre code, nous avons utilisé NB\_BIT = 4. Cette variable permet de définir combien de bit de l’image secrète nous allons cacher dans l’image cachette. Par exemple si nous choisissons NB\_BIT = 7 nous avons :



Nous pouvons voir que l’image secrète est plus nette une fois décodé mais que l’image résultante laisse voir l’image qu’elle est censée cacher.

Inversement, si nous mettons NB\_BIT à 2 nous avons :



Ici, il est impossible de dire que l’image résultante contient un secret, mais en contrepartie, l’image secrète une fois décodé à une qualité moindre.

Ainsi, le choix du nombre de bit faible dépend de l’utilisation que l’on veut faire de la stéganographie.

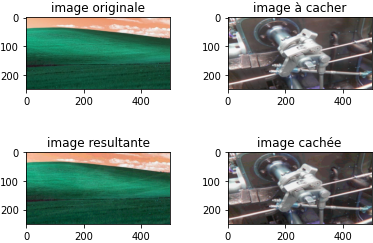
Dans le cas où elle sert pour faire passer des messages secrets qui ne doivent pas être découvert, il est préférable d’utiliser un nombre de bit faible, quitte à réduire la qualité de l’image secrète. A l’inverse, si le but est de garder une bonne qualité après décodage, voire d’avoir un mélange visible des deux images, il faut privilégier un nombre de bit élevé.

De manière générale, 4 bits semblent être un bon compromis puisque cela permet de conserver une bonne qualité sans trahir la présence d’un secret dans l’image cachette.

# Stéganographie sur image en couleur

Le principe pour la stéganographie d’une image en couleur est similaire au principe de la stéganographie d’une image en niveau de gris.

Au lieu de simplement modifier un bit sur un pixel, nous modifions les bits de chaque couleur avec le même procédé ce qui nous donne ce genre de résultat :



Grâce à l’utilisation de la fonction map, notre code supporte à la fois les images en niveau de gris et les images couleurs.