Module: Réseaux II

PROJET TP:

Protocole Graphique

- Réalisé par: FALEK Mahdi et MERIMECHE Walid

L3 Informatique 15 mai 2023

Table des matières

VI. Introduction

Dans le cadre de notre projet de réseau, nous avons été chargés de concevoir un protocole de communication graphique similaire au QR Code. Dans un premier temps, nous avons mené des recherches sur les différents types de codes de communication graphique existants. Parmi ces derniers, le Code-barres a immédiatement retenu notre attention. Nous avons donc décidé de nous inspirer de ce code pour développer le nôtre. Toutefois, nous avons décidé de transmettre l'information le long d'un rectangle plutôt que sous forme de barres.

VII. Programmation

Nous avons choisi d'utiliser Python (version 3) pour ce projet de programmation car c'est l'un des langages que nous maîtrisons le mieux. De plus, Python est un langage très polyvalent qui offre une grande liberté et bénéficie d'une communauté très active. Cette décision nous a permis d'être sûrs de ne pas rencontrer de blocages au cours du projet. Nous avons également utilisé plusieurs bibliothèques, notamment pour la partie graphique, qui doivent être installées sur l'ordinateur avant d'exécuter le programme.

VIII. Fonctionnement

Dans un premier temps, nous avons choisi d'utiliser la table ASCII pour coder les caractères à transmettre. Pour cela, nous avons converti l'information en séquence de bits binaires, que nous avons ensuite stockés sous forme d'un tableau. Cependant, étant donné que notre protocole nécessite une certaine sécurité, nous avons décidé d'utiliser le code de Hamming. Ce code est plus efficace pour détecter et corriger les erreurs, ce qui est particulièrement important dans notre cas car notre protocole est plus facile à lire lorsqu'il y a peu d'informations. Le code de Hamming ajoute des bits de contrôle au code d'origine de l'information à transmettre afin de vérifier les erreurs et de les corriger lors de sa lecture par le récepteur.

3.1 Convert_to_bit ()

Cette fonction est utilisée pour convertir un message texte en format binaire. Elle le fait en itérant sur chaque caractère de la chaîne, en convertissant la représentation ASCII de ce caractère en binaire, et en ajoutant les bits résultants à une liste.

```
def convert_to_bit(msg):
    return [bit for char in msg for bit in bin(ord(char))[2:]]
```

3.2 Code_hamming()

La fonction code_hamming() effectue la tâche principale du codage de Hamming. Elle accepte une liste de bits et renvoie une liste qui comprend à la fois les bits d'origine et des bits de parité supplémentaires, placés à des positions qui sont des puissances de 2. Ces bits de parité sont calculés de telle sorte que, pour n'importe quel groupe de bits comprenant un bit de parité à la position n et les n-1 bits suivants, la somme des bits de ce groupe soit paire.

```
def code_hamming(bits):
    n = len(bits)
    m = next(i for i in range(n + 1) if 2 ** i >= n + i + 1)
    code = [0] * (n + m)
    j = 0

    for i in range(1, len(code) + 1):
        if not (i & (i - 1)):  # if i is a power of 2
            continue
        code[i - 1] = int(bits[j])
        j += 1

    for i in range(1, len(code) + 1):
        if (i & (i - 1)):  # if i is not a power of 2
            continue
        code[i - 1] = sum(int(code[j]) for j in range(i - 1, len(code), 2 * i)) % 2

    return code
1 usage
```

3.3 Code graphique avec image:

La fonction **codage_with_image** génère une image avec des rectangles rouges dessinés dessus, en fonction des données d'entrée sous forme d'une liste d'entiers **tab** et d'une image spécifiée par son chemin.

L'image est chargée à l'aide de la bibliothèque PIL, convertie en un tableau NumPy et affichée à l'aide de la bibliothèque matplotlib. Un rectangle rouge est dessiné au centre de l'image avec une taille de **p** x **p** pixels (**p** est initialisé à 2).

Un autre rectangle rouge est ensuite dessiné avec une taille légèrement plus grande (**p + j*0.1** x **p + j*0.1** pixels) et déplacé légèrement vers la gauche et vers le haut du centre de l'image (**j** est initialisé à 1). La valeur de **j** est ensuite incrémentée à 2.

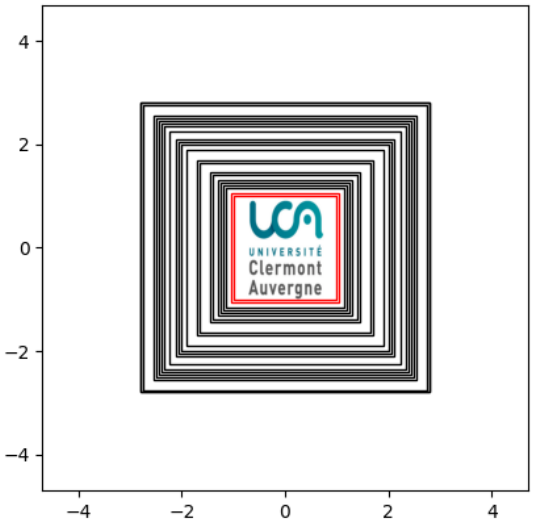
Ensuite, une boucle est exécutée pour chaque élément de la liste d'entrée **tab**. Pour chaque valeur entière positive dans **tab**, un rectangle rouge est dessiné avec une taille légèrement

plus grande que la précédente, la taille augmentant linéairement avec l'indice de boucle **j**. L'indice de boucle **j** est incrémenté après chaque itération.

Enfin, les limites des axes X et Y sont définies en fonction de la valeur maximale de **j**, le rapport d'aspect du graphique est défini à 1, et l'image est enregistrée dans un fichier nommé 'output_with_image.png' et affichée à l'aide de la méthode **show()** du module **matplotlib.pyplot**.

En résumé, ce code génère une image avec des rectangles rouges dessinés dessus, où la taille et la position des rectangles dépendent des données d'entrée dans **tab**.

```
def codage_with_image(tab, image_path):
    p = 2
    X = 0
    Y = 0
    j = 1

    figure, ax = plt.subplots()

    img = Image.open(image_path)
    img = np.array(img)

    rectangle = patches.Rectangle((X - p/2, Y - p/2), p, p, fill=False, color='red')
    ax.add_patch(rectangle)

    ax.imshow(img, extent=[X - p/2, X + p/2, Y - p/2, Y + p/2])

    rectangle = patches.Rectangle((X - (p + j * 0.1)/2, Y - (p + j * 0.1)/2), p + j * 0.1, p + j * 0.1, fill=False, color='red')
    ax.add_patch(rectangle)
    j = 2

    for i in tab:
        if (i > 0):
            rectangle = patches.Rectangle((X - (p + j * 0.1)/2, Y - (p + j * 0.1)/2), p + j * 0.1, p + j * 0.1, fill=False)
            ax.add_patch(rectangle)
        j = j + 1

    plt.xlim(-0.5 - 0.1 * j, 0.5 + j * 0.1)
    plt.ylim(-0.5 - 0.1 * j, 0.5 + j * 0.1)
    ax.set_aspect(1)
    ax.add_patch(rectangle)
    plt.savefig('output_with_image.png')
```



3.4 Code graphique sans image:

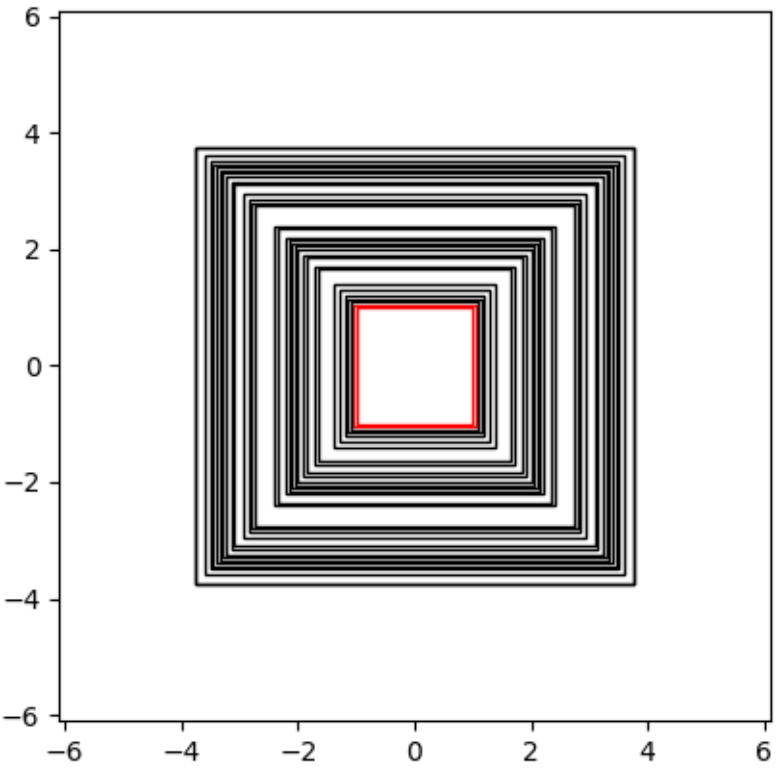
La fonction **codage_without_image** génère une image à partir d'un tableau d'entiers donné en entrée. Chaque entier du tableau correspond à un carré à ajouter à l'image, en fonction de sa valeur.

Le code commence par initialiser des variables pour la position et la taille des carrés. Il crée ensuite un objet **figure** et **axes** avec **subplots()** de la bibliothèque Matplotlib.

Deux rectangles rouges sont dessinés pour les premiers bits d'informations à l'aide de la fonction **patches.Rectangle()** qui crée des rectangles. Ensuite, le code parcourt le tableau et dessine un rectangle pour chaque élément qui a une valeur supérieure à 0. La taille de chaque rectangle est déterminée en fonction de sa position dans le tableau.

Finalement, le code configure les limites et l'aspect de l'image, ajoute un patch supplémentaire à l'axe et sauvegarde l'image sous forme de fichier PNG avec **savefig()**, puis l'affiche avec **show()**. Le nom de l'image de sortie est 'output_without_image.png'.

```python
def codage_without_image(tab):
    p = 2
    X = 0
    Y = 0
    j = 1

    figure, ax = plt.subplots()

    rectangle = patches.Rectangle((X - p/2, Y - p/2), p, p, fill=False, color='red')
    ax.add_patch(rectangle)

    rectangle = patches.Rectangle((X - (p + j * 0.1)/2, Y - (p + j * 0.1)/2), p + j * 0.1, p + j * 0.1, fill=False, color='red')
    ax.add_patch(rectangle)
    j = 2

    for i in tab:
        if (i > 0):
            rectangle = patches.Rectangle((X - (p + j * 0.1)/2, Y - (p + j * 0.1)/2), p + j * 0.1, p + j * 0.1, fill=False)
            ax.add_patch(rectangle)
        j = j + 1

    plt.xlim(-0.5 - 0.1 * j, 0.5 + j * 0.1)
    plt.ylim(-0.5 - 0.1 * j, 0.5 + j * 0.1)
    ax.set_aspect(1)
    ax.add_patch(rectangle)
    plt.savefig('output_without_image.png')
    plt.show()
```

IIII Ameliorations possibles:

- Codage couleur : Pour rendre la visualisation plus intuitive, différentes couleurs pourraient être utilisées pour représenter les bits 0 et 1. Cela pourrait être particulièrement utile dans la visualisation basée sur des images, où il peut être difficile de distinguer la présence et l'absence de rectangles.

- Visualisation interactive : Actuellement, les visualisations sont statiques. En utilisant une bibliothèque de visualisation interactive, telle que Bokeh ou Plotly, il serait possible de rendre les visualisations plus attrayantes. Par exemple, survoler un rectangle pourrait afficher le bit qu'il représente, et cliquer dessus pourrait fournir plus d'informations sur sa position dans le code et le caractère dont il fait partie.

V Conclusion :

Pour conclure, j’ai bien aimé travailler sur ce projet car la vision de sa progression est très motivante et satisfaisante. J’ai pu découvrir des bibliothèques inconnues et améliorer mon niveau dans le langage Python. J’ai également été très curieux sur le domaine du réseau et de la transmission de données mais aussi du lien avec les Mathématiques pour la sécurisation de l’envoie et la redondance (Code Hamming).