Protocole Graphique V2 : Rapport Technique

David Pereira
Sophie Wodey

16 mai 2025

Résumé

Table des matières

Chapitre 1

Introduction

L'objectif de ce projet est de concevoir et d'implémenter un protocole de communication graphique original, inspiré des principes du QR code, dans le cadre du module Réseaux 2. Le protocole doit permettre de transmettre un message texte quelconque sous forme d'une image matricielle, tout en assurant la robustesse face aux altérations possibles lors de l'impression, du scan ou de la transmission de l'image.

Le cahier des charges impose de définir une structure de matrice adaptée, d'intégrer des mécanismes de détection et de correction d'erreurs, et de fournir des informations spécifiques au protocole (version, taille du message, codage des couleurs, etc.). Le projet doit également permettre de localiser la matrice dans une image plus large, d'estimer la taille des cellules, de retrouver l'orientation, et de garantir la fiabilité du décodage même en présence de bruit ou de rotations.

Pour répondre à ces exigences, nous avons développé un protocole graphique structuré autour de plusieurs zones fonctionnelles : motifs de repérage (Finder Patterns), motifs de synchronisation (Timing Patterns), zone de métadonnées, zone de données protégées par code correcteur d'erreurs (ECC), et patches de calibration des couleurs. L'ensemble du pipeline, de l'encodage du texte à la génération de l'image, puis du décodage de l'image au texte, a été conçu pour être modulaire, extensible et robuste.

Ce rapport détaille la conception du protocole, les choix technologiques, les algorithmes mis en œuvre, les tests réalisés, ainsi que les résultats obtenus et les perspectives d'amélioration.

Chapitre 2

Spécifications et conception du protocole

2.1 Cahier des charges et contraintes

Le protocole graphique devait répondre à plusieurs exigences définies par le sujet :

- Permettre la transmission d'un message texte quelconque sous forme d'une image matricielle.
- Intégrer des informations spécifiques au protocole : version, taille du message, codage des couleurs, etc.
- Assurer la robustesse face aux altérations possibles (variation de taille, bruit, rotation, perspective, altération des couleurs).
- Permettre la localisation automatique de la matrice dans une image plus large (grâce à des motifs de repérage).
- Fournir des mécanismes de détection et de correction d'erreurs (ECC).
- Prendre en compte la possibilité d'ajouter un logo sans compromettre la décodabilité (optionnel).

2.2 Choix technologiques

Le projet a été développé en Python, principalement pour sa rapidité de prototypage et la richesse de son écosystème scientifique. Les bibliothèques suivantes ont été utilisées :

- **Pillow** : pour la manipulation et la génération d'images.
- **numpy** : pour les opérations sur les matrices et le traitement des données.
- **reedsolo** : pour l'implémentation du code correcteur d'erreurs Reed-Solomon.
- **argparse** : pour la création de l'interface en ligne de commande (CLI).

Ce choix permet d'assurer la portabilité, la clarté du code, et la facilité de mise en place de tests unitaires.

2.3 Structure générale de la matrice graphique

La matrice graphique est une grille carrée de dimension fixe (par exemple 35x35 cellules pour la version standard). Chaque cellule encode une paire de bits, représentée par

une couleur parmi quatre possibles (blanc, noir, rouge, bleu). Cette organisation permet d'augmenter la capacité d'information par rapport à un codage binaire classique.

La matrice est divisée en plusieurs zones fonctionnelles, chacune ayant un rôle précis dans le protocole :

— Motifs de repérage (Finder Patterns)

— Motifs de synchronisation (Timing Patterns)

— Zone de métadonnées

— Zone de données et de correction d'erreurs (ECC)

— Patches de calibration des couleurs (CCP)

— (Optionnel) Zone réservée à un logo

2.4 Description des zones de la matrice

2.4.1 Finder Patterns (FP)

Les Finder Patterns sont placés dans trois coins de la matrice (haut-gauche, haut-droit, bas-gauche). Ils sont constitués de motifs concentriques de couleurs distinctes, permettant une détection fiable de la position, de l'orientation et de l'échelle de la matrice lors du décodage. Chaque coin possède une couleur centrale unique pour lever toute ambiguïté sur l'orientation.

2.4.2 Timing Patterns (TP)

Les Timing Patterns sont des lignes et colonnes alternant deux couleurs, placées entre les Finder Patterns. Ils servent de repères pour l'alignement précis de la grille et facilitent la correction des déformations lors du décodage.

2.4.3 Zone de métadonnées

Cette zone regroupe les informations nécessaires au décodage : version du protocole, niveau de correction d'erreur, longueur du message, clé de chiffrement, etc. Les métadonnées sont protégées contre les erreurs par une redondance ou un code correcteur simple.

2.4.4 Zone de données/ECC

La plus grande partie de la matrice est dédiée à l'encodage du message utilisateur et des bits de correction d'erreur (ECC). Le message est d'abord converti en bits, éventuellement chiffré, puis protégé par un code correcteur (checksum ou Reed-Solomon) avant d'être placé dans la matrice.

2.4.5 Calibration Color Patches (CCP)

Des patches de couleurs pures sont placés à des emplacements fixes pour permettre la calibration dynamique des couleurs lors du décodage. Cela garantit la robustesse du protocole face aux variations d'impression ou de capture.

2.4.6 Zone logo (optionnelle)

Le protocole prévoit la possibilité de réserver une zone centrale pour l'insertion d'un logo. Cette zone est exclue du codage des données et signalée dans les métadonnées, afin de ne pas compromettre la décodabilité du message.

Chapitre 3

Algorithmes et pipeline de traitement

3.1 Encodage (texte → image)

L'encodage d'un message texte en image suit un pipeline structuré en plusieurs étapes :

1. **Conversion du texte en bits** : Le message utilisateur est converti en une séquence de bits selon l'encodage UTF-8. Un éventuel padding est ajouté pour atteindre la longueur requise.
2. **Chiffrement XOR** : Pour renforcer la sécurité et la diversité des motifs, un chiffrement simple par XOR est appliqué sur les bits du message à l'aide d'une clé générée aléatoirement ou fournie par l'utilisateur.
3. **Ajout de l'ECC** : Un code correcteur d'erreurs (ECC) est calculé sur les bits chiffrés. Selon la configuration, il s'agit soit d'un simple checksum, soit d'un code Reed-Solomon pour une robustesse accrue.
4. **Préparation des métadonnées** : Les informations nécessaires au décodage (version, longueur, clé, niveau d'ECC, etc.) sont assemblées et protégées par redondance.
5. **Placement dans la matrice** : Les bits des métadonnées, du message chiffré et de l'ECC sont placés dans la matrice selon un ordre défini, en respectant les zones réservées (FP, TP, CCP, logo).
6. **Génération de l'image** : La matrice de bits est convertie en une image couleur, chaque paire de bits étant associée à une couleur spécifique. Une marge blanche (quiet zone) est ajoutée autour de la matrice pour faciliter la détection.

Chaque étape est implémentée dans un module distinct, ce qui facilite la maintenance, les tests et l'évolution du protocole.

3.2 Décodage (image → texte)

Le décodage d'une image vers le texte original suit le pipeline inverse :

1. **Détection des motifs de repérage (FP)** : Les Finder Patterns sont localisés dans l'image pour déterminer la position, l'orientation et l'échelle de la matrice.
2. **Calibration des couleurs (CCP)** : Les patches de calibration sont utilisés pour ajuster dynamiquement la correspondance entre couleurs et bits, compensant les variations dues à l'impression ou à la capture.
3. **Extraction de la matrice de bits** : La grille de cellules est extraite de l'image, chaque cellule étant convertie en bits selon la calibration.

4. **Décodage des métadonnées** : Les bits des métadonnées sont lus et interprétés pour obtenir les paramètres nécessaires au décodage (longueur, clé, ECC, etc.).
5. **Vérification et correction ECC** : Les bits du message chiffré sont vérifiés et, si possible, corrigés à l'aide de l'ECC.
6. **Déchiffrement et récupération du texte** : Le message est déchiffré par XOR, puis reconverti en texte UTF-8 en supprimant le padding éventuel.

Ce pipeline garantit la robustesse du décodage, même en présence de rotations, de bruit ou de légères déformations. La modularité du code permet d'adapter ou d'améliorer chaque étape indépendamment.

Chapitre 4

Robustesse et tests

4.1 Robustesse à la rotation, bruit, perspective

Le protocole a été conçu pour garantir la robustesse du décodage dans des conditions variées. Plusieurs mécanismes ont été intégrés :

- **Finder Patterns différenciés** : Les motifs de repérage avec couleurs centrales uniques permettent de détecter l'orientation de la matrice et de corriger automatiquement les rotations à 90°, 180° ou 270°.
- **Quiet zone et marges** : Une bordure blanche entoure la matrice pour l'isoler du fond et faciliter la détection automatique.
- **Timing Patterns** : Les lignes et colonnes de synchronisation aident à corriger les déformations locales et à ajuster précisément la grille de lecture.
- **Calibration dynamique des couleurs** : Les patches de calibration (CCP) permettent d'adapter le décodage aux variations d'impression ou de capture (lumière, contraste, etc.).
- **Correction d'erreurs (ECC)** : L'utilisation d'un code correcteur (checksum ou Reed-Solomon) permet de détecter et corriger un certain nombre d'erreurs dans les données.

4.2 Stratégies de test

La robustesse du protocole a été validée par une série de tests :

- **Tests unitaires** : Chaque module (encodage, décodage, placement des motifs, ECC, etc.) a été testé individuellement à l'aide de jeux de données contrôlés.
- **Tests d'intégration** : Le pipeline complet (texte → image → texte) a été testé pour vérifier la fidélité de la transmission.
- **Tests sur images altérées** : Des images générées ont été modifiées (rotation, ajout de bruit, altération des couleurs, déformation) pour évaluer la robustesse du décodage.
- **Cas limites** : Des tests ont été réalisés avec des messages de longueur maximale, des clés aléatoires, et différents niveaux d'ECC.

4.3 Résultats des tests

Les tests ont montré que le protocole permet de récupérer fidèlement le message original dans la grande majorité des cas, y compris après rotation ou ajout de bruit modéré. La détection des motifs de repérage et la calibration des couleurs assurent une bonne tolérance aux variations d'impression ou de capture. Les limites observées concernent principalement les cas de dégradation extrême (fort bruit, masquage de plusieurs motifs de repérage, ou altération importante des couleurs), pour lesquels le décodage peut échouer ou nécessiter une intervention manuelle.

Chapitre 5

Interface utilisateur et utilisation

5.1 Description de la CLI

Le programme principal propose une interface en ligne de commande (CLI) permettant d'encoder un message texte en image ou de décoder une image pour retrouver le message original. La CLI a été conçue pour être simple d'utilisation et flexible.

Les principales commandes sont :

— `encode` : pour générer une image à partir d'un message texte.

— `decode` : pour extraire le message d'une image générée par le protocole.

Chaque commande accepte plusieurs options, par exemple :

— `-message` : le texte à encoder (pour `encode`)

— `-output` : le nom du fichier image de sortie

— `-ecc` : le niveau de correction d'erreur souhaité (en pourcentage)

— `-key` : la clé de chiffrement XOR (optionnelle)

— `-logo` : chemin vers un logo à insérer (optionnel)

— `-size` : taille de la matrice (ex : V2_S, V2_M)

— `-input` : le fichier image à décoder (pour `decode`)

5.2 Exemples d'utilisation

Voici quelques exemples d'utilisation de la CLI :

— Encodage d'un message simple :

```
python src/main.py encode --message "Bonjour" --output code.png --ecc 20
```

— Décodage d'une image :

```
python src/main.py decode --input code.png
```

— Encodage avec clé personnalisée et logo :

```
python src/main.py encode --message "Test" --output code2.png --key 10101010
```

5.3 Limitations connues et conseils d'utilisation

Le protocole fonctionne de manière fiable dans la plupart des cas standards. Cependant, certaines limitations subsistent :

— La robustesse au bruit ou à la déformation extrême dépend du niveau d'ECC choisi et de la qualité de l'impression ou du scan.

— L'ajout d'un logo réduit la capacité utile de la matrice et peut compliquer le décodage si le logo n'est pas bien contrasté.

— Il est recommandé d'utiliser des images de bonne résolution et d'éviter les compressions destructives (JPEG).

— En cas d'échec du décodage, il peut être utile de vérifier l'orientation de l'image ou d'augmenter le niveau d'ECC.

Chapitre 6

Analyse critique et perspectives

Le protocole développé présente plusieurs points forts. Sa structure modulaire facilite la maintenance, l'ajout de nouvelles fonctionnalités et la réalisation de tests unitaires. L'utilisation de motifs de repérage différenciés, de patches de calibration et d'un code correcteur d'erreurs assure une bonne robustesse face aux altérations courantes (rotation, bruit, variations de couleur). Le choix de Python et de bibliothèques standards permet une portabilité et une clarté du code appréciables.

Cependant, certaines limites subsistent. La robustesse du décodage dépend fortement de la qualité de l'image (résolution, contraste, absence de compression destructrice). Les cas de dégradation extrême (fort bruit, masquage de plusieurs motifs de repérage, altération importante des couleurs) peuvent conduire à l'échec du décodage. L'ajout d'un logo, bien que prévu, réduit la capacité utile et peut compliquer la détection si le contraste n'est pas suffisant.

Plusieurs pistes d'amélioration sont envisageables :

— Intégration d'algorithmes de compression pour augmenter la capacité utile de la matrice.

— Développement de nouveaux schémas de correction d'erreurs plus performants ou adaptatifs.

— Extension du protocole à des matrices de tailles variables ou à des formats non carrés.

— Amélioration de la détection automatique des motifs de repérage et de la correction de perspective.

— Optimisation de l'interface utilisateur et ajout de fonctionnalités avancées (prévisualisation, analyse de robustesse, etc.).

Ce protocole constitue une base solide pour des applications de communication graphique robustes et personnalisables, et peut être enrichi selon les besoins futurs.

Chapitre 7

Conclusion

Ce projet a permis de concevoir et d'implémenter un protocole graphique original, capable de transmettre de manière fiable un message texte sous forme d'image. L'ensemble des étapes, de la définition des spécifications à la réalisation des tests, a été mené en respectant les exigences de robustesse, de modularité et de clarté du code.

Le protocole développé intègre des mécanismes avancés de repérage, de calibration et de correction d'erreurs, assurant une bonne tolérance aux altérations courantes. Les tests réalisés ont confirmé la capacité du système à restituer fidèlement le message dans la plupart des situations usuelles.

Ce travail constitue une base solide pour des applications futures de communication graphique, et pourra être enrichi par l'ajout de nouvelles fonctionnalités ou l'optimisation de certains modules selon les besoins identifiés.

Bibliographie