Reseau 2
PROTOCOLE GRAPHIQUE

Lucas BATTAGLIA
L3 Informatiques

Référent : CHEBBI SAMAR
Responsable d'UE : Patrice LAURENCOT

2024-2025

Table des matières

Table des figures

Liste des tableaux

1 Introduction

Dans le cadre de l'Unité d'Enseignement "Réseau 2" de la Licence 3 Informatique à l'Université Clermont Auvergne, ce projet encadré par Mme Chebbi Samar et M. Patrice Laurencot a pour objectif de concevoir un protocole de communication graphique original.

Inspiré par des systèmes existants comme le QR Code, ce projet propose d'explorer une nouvelle approche de transmission d'informations sous forme graphique. Il s'agit non seulement de définir la structure du message et de son codage, mais également d'assurer la robustesse du protocole face aux erreurs d'impression, aux distorsions et aux modifications d'orientation.

Après avoir étudié le protocole du QR Code, ce rapport présente la création d'un protocole personnel nommé DataMorse, en détaillant sa conception théorique ainsi que son implémentation en Python. L'objectif est de parvenir à une matrice graphique capable d'encoder un message, d'intégrer des mécanismes de correction d'erreurs et de définir des métadonnées spécifiques à ce protocole.

Ce travail comprend ainsi l'analyse du QR Code, la définition du protocole DataMorse, son implémentation technique, suivies d'une conclusion et d'une bibliographie.

2 QRCode (TP1 et 2)

Le QR Code, ou Quick Response Code, est un code-barres en deux dimensions capable de contenir un grand volume d'informations. Contrairement aux codes-barres classiques qui ne stockent les données que dans un seul sens (horizontal), le QR Code les encode à la fois horizontalement et verticalement. Grâce à cette double dimension, il peut contenir plusieurs centaines de caractères dans un carré relativement petit. Son succès vient de sa rapidité de lecture, de sa résistance aux erreurs et de sa facilité d'impression.

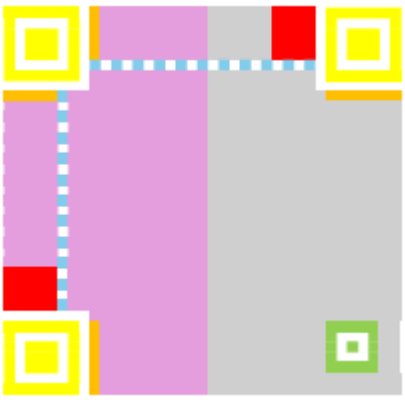

FIGURE 1 – Structure synthétique d'un QR Code

I Le fonctionnement de base

Un QR Code est une grille carrée composée de petits carrés noirs et blancs appelés modules. Chaque module représente un bit : un carré noir vaut 1, un carré blanc vaut 0. Le nombre de modules par côté dépend de la version du QR Code. La version 1, par exemple, mesure 21×21 modules. Plus la version est élevée, plus le QR Code peut contenir de données.

Pour que le QR Code soit lisible, sa structure est toujours la même. Certains modules sont réservés à des éléments de guidage. Ces zones permettent aux lecteurs (comme les caméras de smartphones) de reconnaître le code, de l'orienter correctement, puis de lire l'information.

II Les motifs de positionnement (en jaune)

Dans trois coins du QR Code (pas celui inférieur droit), on trouve de grands carrés noirs et blancs appelés motifs de positionnement. Ils servent à détecter les bords du code, à identifier son orientation, et à faciliter son redressement même s'il est penché ou tordu. Chaque motif de position est formé de sept module par sept avec a l'intérieur un carre blanc de 5 module par 5 contenant lui même un carre noir de 3 par 3!

III Les motifs d'alignement (en vert)

À partir de la version 2, des motifs d'alignement sont ajoutés. Ces petits carrés permettent de corriger les déformations dues à une mauvaise impression ou à une lecture inclinée. Ils sont placés à des emplacements précis dans la grille selon la version du QR Code. Leur présence est essentielle pour garantir une bonne lecture sur de grands QR Codes.

IV La zone de synchronisation (en bleu)

La zone de synchronisation est une ligne de modules alternés en noir et blanc. Elle est placée entre les motifs de position, à la fois horizontalement et verticalement. Elle permet de détecter le rythme des modules et de garantir

que le scanner lit bien chaque ligne avec la bonne régularité. Sans cette synchronisation, la lecture serait souvent erronée.

V La zone silencieuse

Tout autour du QR Code, il doit y avoir une zone blanche vierge appelée zone silencieuse. Elle ne contient aucun module. Cette marge permet de bien séparer le QR Code de tout autre élément graphique ou texte présent autour de lui. Elle est indispensable au bon fonctionnement du décodeur.

VI Le codage des données

Le contenu du QR Code est d'abord transformé en données binaires. Il existe plusieurs modes de codage selon le type de données : le mode numérique (pour les chiffres), le mode alphanumérique (pour les lettres et chiffres), le mode binaire (pour tout type de caractère), et le mode Kanji (pour les caractères japonais). Le message est converti en une suite de bits, selon une table de conversion propre à chaque mode.

VII Le placement des données (en gris)

Les bits de données ne sont pas placés au hasard dans la grille. Ils suivent un chemin de placement en zigzag qui part du coin inférieur droit du QR Code et remonte deux colonnes à la fois, de bas en haut puis de haut en bas, jusqu'à remplir tout l'espace disponible. Ce parcours évite les zones réservées aux motifs et aux métadonnées.

VIII Le système de masquage

Après avoir placé les données, un masque est appliqué. Ce masque est un motif logique qui modifie certains bits pour éviter d'avoir trop de carrés noirs ou blancs consécutifs. Par exemple, si une ligne contient trop de modules noirs, cela peut tromper le lecteur. Le QR Code applique donc l'un des 8 masques définis dans la norme et choisit celui qui donne le meilleur équilibre visuel. Ce masque est indiqué dans une zone spéciale du QR Code (en orange) pour que le lecteur puisse l'annuler avant de lire les données.

Masque	**Condition de masquage**
0	$(i+j) \bmod 2 = 0$
1	$i \bmod 2 = 0$
2	$j \bmod 3 = 0$
3	$(i+j) \bmod 3 = 0$
4	$\left(\left\lfloor \frac{i}{2} \right\rfloor + \left\lfloor \frac{j}{3} \right\rfloor\right) \bmod 2 = 0$
5	$(i \cdot j) \bmod 2 + (i \cdot j) \bmod 3 = 0$
6	$\big(((i \cdot j) \bmod 2 + (i \cdot j) \bmod 3) \bmod 2 = 0\big)$
7	$\big(((i+j) \bmod 2 + (i \cdot j) \bmod 3) \bmod 2 = 0\big)$

TABLE 1 – Les 8 masques des QR Codes et leurs conditions d'application

IX La correction d'erreurs (en violet)

Le QR Code intègre un système de correction d'erreurs très puissant basé sur l'algorithme de Reed-Solomon. Celui-ci ajoute des blocs redondants aux données pour permettre de reconstruire le message même si une partie du QR Code est abîmée ou illisible. Il existe quatre niveaux de correction : L (7 %), M (15 %), Q (25 %) et H (30 %). Plus le niveau est élevé, plus le code résiste aux erreurs, mais il occupe plus d'espace. Les information sont enregistrer dans une zone spéciale du QR Code (en rouge)

X La lecture du QR Code

Lorsqu'un lecteur scanne un QR Code, il commence par repérer les motifs de position. Il utilise ensuite les motifs d'alignement et la zone de synchronisation pour redresser et calibrer la grille. Ensuite, il lit les données

en suivant le chemin prévu, annule le masque appliqué, puis utilise les blocs de correction pour reconstruire les données originales. Enfin, il convertit les bits en texte ou en lien, selon le contenu encodé.

XI Conclusion

Le QR Code repose sur une combinaison très intelligente de géométrie, de codage binaire, de redondance d'information et de correction d'erreurs. Chaque carré a un rôle précis, et chaque étape du processus — depuis l'encodage des données jusqu'au choix du masque — vise à garantir une lecture rapide et fiable. En comprenant bien cette structure, il est tout à fait possible de concevoir et dessiner un QR Code soi-même, soit à la main, soit en le générant par un programme.

3 DataMorse

Nous avons développé un nouveau protocole graphique nommé DataMorse. Le nom "DataMorse" résulte de l'association entre la notion de données ("Data") et l'idée de transmission codée grâce au "Morse". Ce protocole utilise une matrice de triangles pour représenter les informations, avec une approche symbolique plutôt qu'un codage binaire classique. DataMorse a été conçu pour être facilement identifiable sur une feuille imprimée et lisible par une machine, même en cas de déformation ou d'orientation aléatoire.

I Le protocole (TP 2, 3 et 4)

Le protocole intègre des mécanismes de détection et de correction d'erreurs, tout en s'adaptant à la taille du message à transmettre. Principalement noir et blanc pour maximiser sa robustesse, il permet également l'insertion d'un logo en arrière-plan. L'ensemble du projet a été implémenté en Python, en utilisant ses bibliothèques graphiques efficaces et son approche orientée objet.

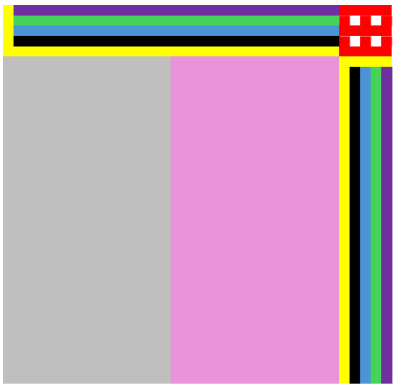

FIGURE 2 – Structure synthétique d'un DataMorse

A Le codage des données

Le DataMorse est une grille carrée composée de petits triangles noirs. Chaque lettre des données à encoder est transformée, selon le code Morse, en un ensemble de tirets et de points. Un ensemble de 3 triangles consécutifs correspond à un tiret, tandis qu'un triangle seul correspond à un point. La taille du dessin est constituée du carré parfait supérieur au nombre de triangles nécessaires à l'encodage de l'ensemble du message. Par exemple, pour un message nécessitant 65 triangles, on aura besoin d'une matrice de 81 (9 par 9) !

Caractère	Morse	Caractère	Morse	Caractère	Morse	Caractère	Morse	Caractère	Morse
A	.-	B	-...	C	-.-.	D	-..	E	.
F	..-.	G	–.	H	....	I	..	J	.—
K	-.-	L	.-..	M	–	N	-.	O	—
P	.–.	Q	–.-	R	.-.	S	...	T	-
U	..-	V	...-	W	.–	X	-..-	Y	-.–
Z	–..	0	——	1	.—-	2	..—	3	...–
4	....-	5	.....	6	-....	7	–...	8	—..
9	—-.	.	.-.-.-	,	–..–	?	..–..	’	.—-.
!	-.-.—.	/	-..-.	(	-.–.	)	-.–.-	&	.-...
:	—...	;	-.-.-.	=	-...-	+	.-.-.	-	-....-
_	..–.-	"	.-..-.	$	...-..-	@	.–.-.		

TABLE 2 – Correspondance entre caractères et code Morse

B Les motifs de positionnement (en rouge)

Contrairement au protocole QRCode, nous avons besoin d'un seul motif de positionnement. Nous le plaçons dans le coin supérieur droit. Le motif de positionnement est un carré de 5 triangles par 5, avec une croix droite à l'intérieur.

Pour éviter les rotations, nous allons vérifier la position de nos triangles, qui doivent pointer vers le haut! Tant que ce n'est pas le cas, nous devons faire pivoter notre image. Une fois les triangles orientés vers le haut, nous devons vérifier le positionnement de notre motif. S'il est à sa place, alors nous avons le bon DataMorse, sinon nous avons un effet miroir ou une transparence.

C La zone silencieuse

Comme pour le QRCode, nous recommandons de garder une zone vide entre le texte, une image et le DataMorse. Cependant, il est possible de la supprimer si nécessaire.

D Le placement des données (en gris)

Les symboles de données ne sont pas placés dans les premières lignes! Nous avons placé les données à partir de la $6^{\text{ème}}$ ligne et jusqu'à la fin, mais en privant chaque ligne des 5 dernières colonnes! Contrairement au QRCode, on lit les données de haut en bas et de gauche à droite!

E Le système de masquage

Après avoir placé les données, un masque est appliqué. Ce masque est une expression logique qui modifie la présence du triangle afin d'éviter de trop grandes zones blanches consécutives. Cela facilite la lecture du DataMorse! Plusieurs masques existent. Afin de connaître le masque utilisé, le programme de décodage lira la première ligne (en violet). Cette ligne contiendra l'identifiant du masque utilisé pour crypter les données!

Masque	Condition logique	Commentaire
1	$(i \bmod 3 + j \bmod 2) \bmod 2 = 0$	Combine des fréquences différentes sur chaque axe.
2	$(i \cdot j) \bmod (i + j + 1) \bmod 2 = 0$	Modulation adaptative à la position, produit et somme.
3	$\left\lfloor \frac{i \cdot j}{i+j+1} \right\rfloor \bmod 2 = 0$	Évite les répétitions linéaires, variation lissée.
4	$(i^2 + 3j^2) \bmod 5 \bmod 2 = 0$	Motif quadratique non-linéaire, évite les symétries simples.
5	$\lfloor \frac{i}{3} \rfloor + j \bmod 2 = 0$	Crée des bandes verticales déphasées.
6	$(i \cdot (j+1)) \bmod 7 \bmod 2 = 0$	Produit désaxé, évite motifs centrés ou répétés.

TABLE 3 – Masques pour le protocole DataMorse

F La correction d'erreurs

Pour ce protocole, nous utilisons deux codes détecteurs et correcteurs d'erreur :

— Pour les données, nous utilisons l'algorithme de Reed-Solomon. Il permet de détecter et corriger des erreurs dans des données. Il fonctionne en ajoutant des symboles redondants (en rose) à un message original. Ces symboles sont générés de manière à pouvoir reconstruire les données originales, même si certaines parties sont corrompues.

  En pratique, Reed-Solomon utilise des polynômes sur un corps fini (généralement $\mathbf{GF}(\mathbf{2^m})$) pour encoder et décoder les données. Lors de la transmission ou du stockage, si des erreurs surviennent, l'algorithme peut détecter et corriger un certain nombre d'erreurs en utilisant les symboles de parité. Plus le nombre de symboles de parité ajoutés est élevé, plus le code peut corriger d'erreurs, mais cela augmente aussi la taille du message.

Les paramètres permettant de décoder les données (la longueur du message en vert, les bits de remplissage en bleu et le nombre de parité en noir) sont stockés sur les $2^{ème}$ (en vert), $3^{ème}$ (en bleu) et $4^{ème}$ (en noir) lignes de paramétrage.

— Pour les lignes de paramétrage, nous utilisons l'algorithme de Hamming. Il permet de corriger des erreurs grâce à des bits de parité (en jaune) ajoutés à un message pour détecter et corriger des erreurs simples. Ces bits vérifient la parité (paire ou impaire) des bits voisins, ce qui permet de repérer les erreurs.

Lors de la réception, l'algorithme vérifie les bits de parité. Si une erreur est détectée (par exemple, un bit de données a été inversé), l'algorithme identifie l'emplacement exact du bit erroné et le corrige. L'algorithme peut corriger une seule erreur à la fois, mais il ne peut pas gérer plusieurs erreurs simultanées.

Il n'a pas besoin de paramètre pour fonctionner, c'est pourquoi nous avons choisi ce code (moins performant) pour crypter nos lignes de paramétrage!

G Ligne de paramétrage

Les lignes de paramétrage permettent de récupérer les informations nécessaires pour décoder les données! Pour être à peu près sûr de pouvoir obtenir les informations, nous créons de la redondance. Les informations sont placées sur les 4 premières lignes et les 4 dernières colonnes. La $5^{ème}$ ligne sert à enregistrer les bits de parité pour le code de Hamming! Les 5 dernières colonnes sont donc une transposition des 5 premières lignes!

Afin de mieux controler la taille de ses informations, se sera les seuls information coder en binaire!

II L'implémentation (TP 3, 4 et 5)

A L'encodage

L'implémentation du protocole DataMorse a été réalisée en plusieurs étapes, intégrant des mécanismes de détection et de correction d'erreurs, tout en s'adaptant à la taille du message à transmettre. L'ensemble du projet a été implémenté en Python, en utilisant ses bibliothèques graphiques efficaces et son approche orientée objet.

L'encodage des données dans le protocole DataMorse se fait en plusieurs étapes :

1. **Transformation du texte en Morse :** Chaque lettre du message est convertie en une séquence de points et de tirets selon le code Morse. Cette transformation est réalisée par la méthode __text _to_morse, qui génère une liste de bits représentant le message.
2. **Système de détection et correction d'erreurs :** Le protocole DataMorse intègre un système de détection et de correction d'erreurs basé sur l'algorithme de Reed-Solomon. Ce système ajoute des symboles de parité aux données encodées, permettant de détecter et de corriger jusqu'a 20 % des erreurs potentielles lors de la transmission ou du stockage du message. Les paramètres de correction sont stockés dans les lignes de paramétrage de la matrice, garantissant que même si une partie du DataMorse est endommagée, le message original peut être reconstruit.
3. **Création de la matrice :** Une fois le message en Morse obtenu, il est transformé en une matrice de triangles. La méthode __morse_to_matrix est utilisée pour créer cette matrice, où chaque triangle représente un bit du message.
4. **Application de masques :** Pour éviter des zones trop uniformes dans la matrice, un système de masquage est appliqué. La classe Masque permet d'appliquer un masque logique qui modifie certains bits de la matrice, améliorant ainsi la robustesse du protocole.
5. **Ajout de l'entête :** Un motif de positionnement est ajouté dans le coin supérieur droit de la matrice pour faciliter la lecture et l'orientation du DataMorse. Ce motif est essentiel pour garantir que le message peut être correctement interprété, même s'il est désorienté. A gauche et en dessous de ce motif sont ajouté les donnée nécessaire au déchiffrage du DataMorse!
6. **Génération de l'image :** La méthode __generate_image crée une image à partir de la matrice binaire. Si un chemin d'image de fond est fourni, cette image est superposée avec une transparence de 75%, permettant d'intégrer un logo ou une autre image de manière esthétique.

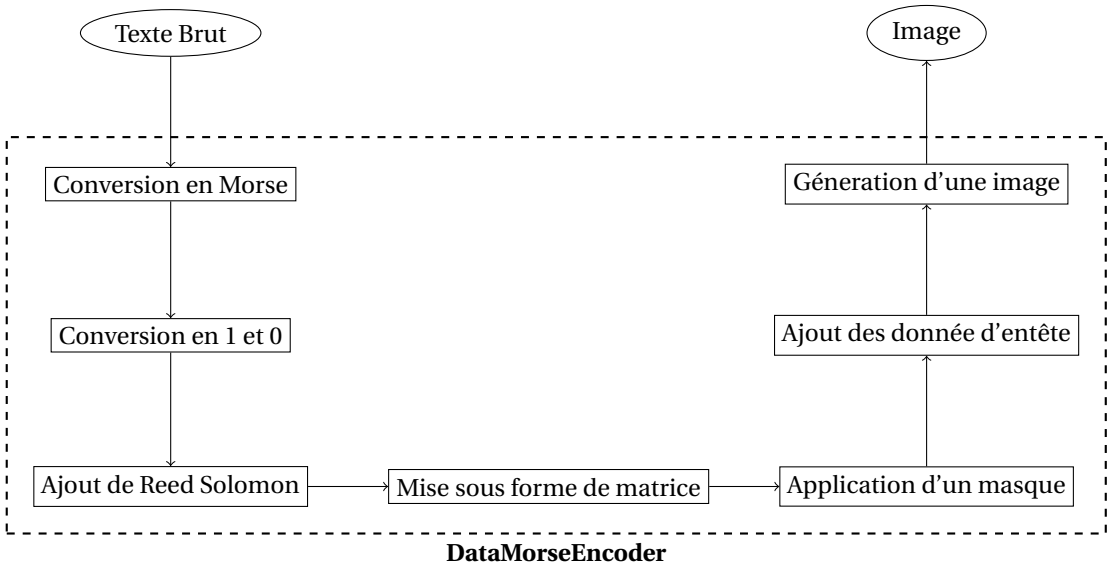

FIGURE 3 – Étapes du processus de d'encodage d'un message au format DataMorse

B Le décodage

Le décodage des messages encodés en DataMorse est une étape cruciale qui permet de récupérer le message original à partir de l'image ou de la matrice. Cette étape est réalisée par la classe DataMorseDecoder, qui se divise en trois sous-classes : le décodeur via la caméra, le décodeur d'image et le décodeur de matrice.

a Décodeur via la caméra

Cette partie n'a pas encore été implémentée! Pour disposer de cette fonctionnalité, il faudrait réussir à réduire considérablement le temps d'exécution du décodage à partir d'une image. L'idée est d'utiliser la caméra pour capturer des images en temps réel et d'appliquer le décodage directement sur ces images. Cela nécessiterait une optimisation des algorithmes de traitement d'image pour garantir une lecture rapide et efficace des données encodées dans le DataMorse.

b Decodeur d'image

Le décodeur d'image est conçu pour traiter des fichiers d'image contenant le DataMorse. Pour lire le DataMorse, nous utilisons la bibliothèque cv2. Voici les étapes clés du processus de lecture :

1. **Chargement de l'image :** La première étape consiste à charger l'image à partir du chemin spécifié. Cela se fait en utilisant la fonction cv2.imread, qui lit l'image et la convertit en un format que nous pouvons manipuler.
2. **Application d'un filtre GaussianBlur :** Pour améliorer la qualité de l'image et réduire le bruit, nous appliquons un filtre GaussianBlur. Ce filtre aide à lisser l'image, ce qui est particulièrement utile pour éliminer les petites imperfections qui pourraient affecter la détection des triangles. La méthode cv2.GaussianBlur est utilisée pour appliquer ce filtre, en spécifiant la taille du noyau et l'écart type.
3. **Conversion en niveaux de gris :** Après le flou, l'image est convertie en niveaux de gris à l'aide de cv2.cvtColor. Cette conversion simplifie le traitement ultérieur, car nous n'avons besoin de travailler qu'avec une seule couche de couleur.
4. **Seuil de binarisation :** Pour transformer l'image en une matrice binaire, nous appliquons un seuil à l'image en niveaux de gris. La méthode cv2.threshold est utilisée pour définir un seuil, où les pixels au-dessus de ce seuil sont convertis en blanc (1) et ceux en dessous en noir (0). Cela permet de créer une image binaire qui représente clairement les triangles.

5. **Extraction des contours :** Une fois l'image binaire obtenue, nous utilisons cv2.findContours pour détecter les contours des formes présentes dans l'image. Cette étape est cruciale pour identifier les triangles qui composent le DataMorse.

6. **Récupération des triangles :** La méthode __recup_triangle est ensuite utilisée pour extraire les triangles à partir des contours détectés. Chaque triangle est analysé pour déterminer s'il représente un bit de données valide.

7. **Filtrage des points isolés :** Pour garantir la précision du décodage, la méthode __filter_isolated_points est utilisée pour éliminer les points isolés qui pourraient fausser le message. Cela permet d'améliorer la fiabilité du décodage en s'assurant que seuls les points pertinents sont pris en compte.

8. **Conversion du Morse en texte :** Une fois la matrice binaire obtenue, la méthode __morse_to_text convertit la séquence de Morse en texte lisible, permettant ainsi de récupérer le message original. Cette conversion est essentielle pour rendre le message compréhensible pour l'utilisateur final.

Ensuite on appelle la classe pour le décodage à partir de la matrice !

Difficulté rencontrée

Bien que cv2 est une bibliothèque qui est relativement simple, n'ayant pas eu de cours sur la gestion d'images dans notre cursus, nous avons eu un peu de mal à analyser l'image en cas de photo, de déformation, de rotation, etc.

c Décodeur de matrice

Le décodeur de matrice est une sous-classe permettant de décoder des messages à partir de matrices binaires. Cette approche est utile lorsque le DataMorse est déjà sous forme matricielle, notamment après traitement d'image. Voici les étapes clés du processus :

1. **Gestion de l'orientation :** Il est crucial de corriger l'orientation de la matrice, car le DataMorse peut subir des rotations (90°, 180°). La position des motifs de repérage permet d'identifier cette orientation. Comme vu en TP, les inversions par symétrie ne sont pas traitées, mais elles pourraient l'être en suivant les principes décrits dans la partie protocole.

2. **Extraction de l'en-tête :** L'en-tête contient des informations essentielles comme le masque appliqué ou les paramètres de correction d'erreurs. Ces métadonnées guident le traitement du reste de la matrice.

3. **Suppression du masque :** Le masque est un motif logique utilisé à l'encodage pour éviter les zones trop homogènes. Il est retiré afin de retrouver les données brutes.

4. **Passage en chaîne :** Les données de la matrice sont converties en une chaîne linéaire, ce qui facilite la correction d'erreurs et la conversion en texte.

5. **Correction des données (Reed-Solomon) :** Grâce aux symboles de parité ajoutés lors de l'encodage, l'algorithme de Reed-Solomon permet de corriger d'éventuelles erreurs, assurant l'intégrité du message.

6. **Décodage du Morse :** Enfin, la séquence obtenue est traduite du Morse vers un texte lisible, restituant le message original à l'utilisateur.

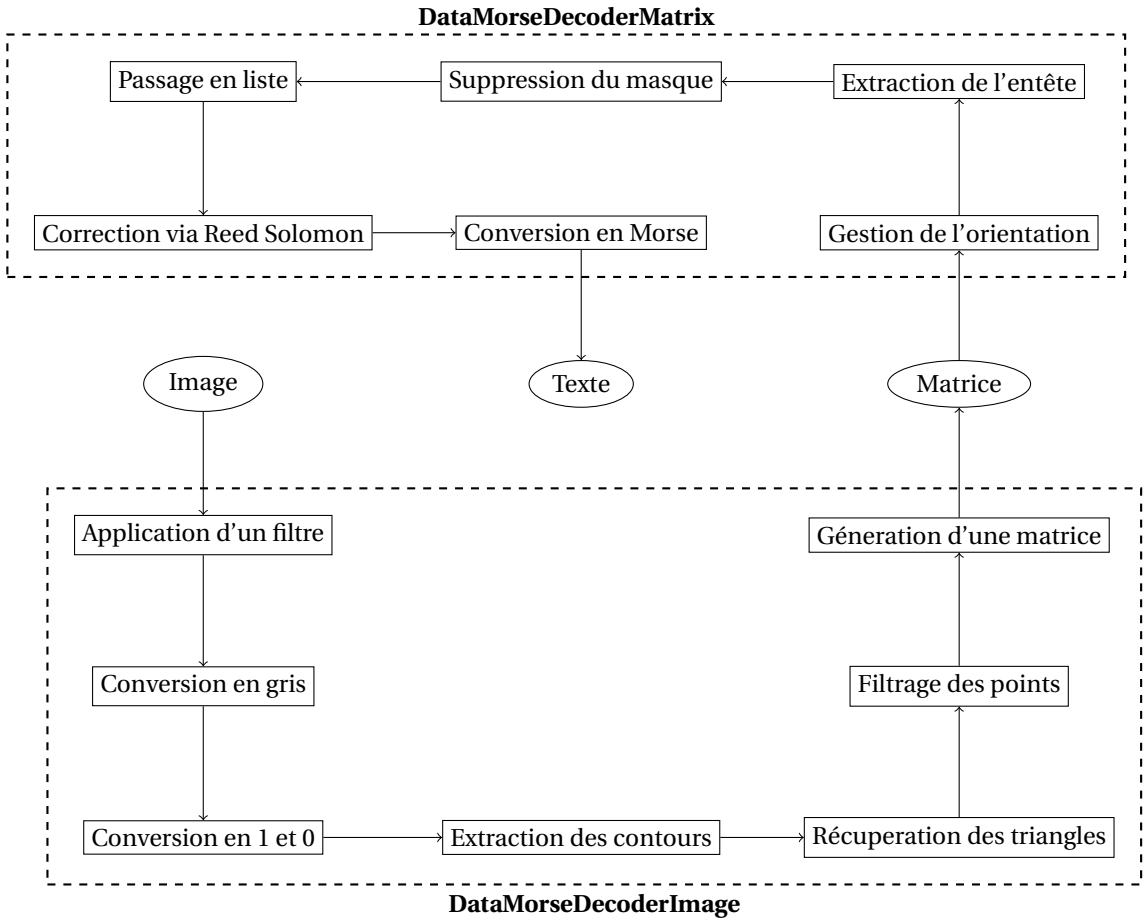

FIGURE 4 – Étapes du processus de décodage d'un message au format DataMorse

C Limites

Bien que théoriquement robuste, ce protocole présente certaines limites :

1. **Taille du message :** Bien qu'il soit possible de créer des DataMorse de plus de 500 caractères, cela pose des problèmes au décodage. Le programme devient très gourmand en mémoire, et le nombre de triangles nécessaires au-delà de 300 caractères le rend trop lourd à traiter. Ce seuil peut varier selon la puissance de la machine utilisée.
2. **Gestion de l'image :** La mise en œuvre de la gestion d'image avec cv2, décrite dans la partie protocole, s'avère trop complexe à notre niveau. Ainsi, seules les rotations sont prises en compte — les inversions par symétrie ou les vues à travers un miroir ne sont pas gérées.
3. **Détection par caméra :** En l'état actuel, la détection par caméra est trop lente pour être exploitable. Une amélioration significative des performances d'exécution serait nécessaire avant d'envisager cette fonctionnalité.
4. **Présence de logos perturbateurs :** Un logo contenant des triangles pourrait interférer avec le DataMorse, en provoquant des erreurs de décryptage. Bien que le blanchiment appliqué à l'image soit censé neutraliser ces éléments, il n'est pas toujours suffisant.

4 Conclusion

Ce projet nous a permis de concevoir et d'implémenter un protocole de communication graphique original, le DataMorse, en nous inspirant des principes du QR Code tout en introduisant des mécanismes spécifiques, comme un codage par triangles et une ligne de paramétrage.

Au-delà de l'aspect technique, ce travail nous a offert une expérience concrète de conception de protocole : de la théorie (structures, masquage, correction d'erreurs) à la pratique (implémentation Python, tests et limites). Il a également révélé les défis liés au traitement d'images, à la gestion mémoire ou encore à la reconnaissance optique via caméra — autant de pistes d'amélioration possibles.

Si certaines contraintes (notamment de performance ou de robustesse à la rotation et aux interférences graphiques) limitent aujourd'hui le déploiement pratique du DataMorse, ce protocole constitue une base solide et personnalisable pour l'exploration de nouvelles formes de communication graphique.

5 Bibliographie

— Wikipedia
— Video "Comment fonctionne un QR code ? Ça se falsifie ?"
— ChatGPT pour la correction des fautes et la reformulation des documentations et du rapport + la comprehension des erreurs python
— Documentation d'OpenCV
— Documentation ReedSolomon
— Documentation PIL
— Différent forum (certain bout de code en sont fortement inspiré)