Indications TP

Apparemment, beaucoup se posent beaucoup de questions à propos du TP, donc voilà quelques pistes...

Préambule :

- Je ne suis pas à l'origine du sujet ;
- Je ne le corrigerai pas ;
- Tout ce que j'écris ici vient d'observations *extérieures* du sujet, comme si j'étais un étudiant qui découvrait le TP.

Prenez donc ce qui est écrit ici avec... pincettes et esprit critique.

"Le TP n'a rien à voir avec du réseau"

Personnellement, je vois le(s) lien(s) suivant(s) :

1. Les deux servent à transmettre des informations (à distance, en plus) :
   - Le rôle (simplifié) des **couches basses** – *qui sont le sujet du cours, je précise, au cas où* – est de transmettre des informations (souvent représentées sous forme binaire en informatique) entre deux machines distantes partageant un "medium" physique (ou un segment, mais ce n'est pas le sujet ici) ;
   - Le rôle (simplifié) des "codes graphiques"/"protocoles de communication graphique" (dont le QR code est ***UN*** exemple) est de transmettre des informations via affichage & scan.
2. ***Problématique commune*** : encoder des données binaires (en général) sur un médium physique : conducteurs, fibres, ondes radio d'un côté, support d'affichage de l'autre ;
   - D'un côté du canal, on "**injecte**" les informations d'une manière ou d'une autre : on force le potentiel électrique à varier, on injecte de la lumière *vs.* on imprime des "pixels", on affiche une image ;
   - Le but du jeu c'est qu'on arrive à les "**lire**" de l'autre côté du canal : on regarde le potentiel, on "observe" la lumière *vs.* on scanne l'image et on l'analyse.
3. ***Problématique commune*** : il faut gérer le bruit (le rapport signal/bruit plutôt) :
   - Interférences, bruit thermique, etc. et atténuation dans les câbles, fibres, ondes...
   - Flou généré par l'optique, couleurs fausses, reflets, définition faible, angles de prise de vue, taches de café sur la feuille, code en partie caché par un objet... beaucoup de choses font que l'image scannée n'est pas parfaitement équivalente à l'image du code telle qu'on l'a produite. Si un protocole de communication graphique tolère mal ça, alors on ne pourra le scanner que :
     i. parfaitement en face ;

ii. avec un éclairage parfait ;
iii. une impression parfaite (pas de bavures...) ;
iv. etc.

Il sera donc inutilisable, clairement. Le QR code est bien mieux conçu ; faites donc des tests pour voir à quel point le QR code tolère les erreurs : penchez le support, faussez certaines "cellules", cachez une partie du code...

En fait, tout protocole de communication doit gérer des aléas physiques, y compris les protocoles graphiques.

"Est-ce qu'il faut refaire le QR code ?"

Non. Dans le sujet, il est écrit : "*Vous devrez concevoir un nouveau protocole de A à Z. Vous pouvez vous inspirer du QR-CODE mais devrez proposer vos propres solutions aux différents problèmes posés.*". C'est assez clair, non ?

Si vous cherchez (même juste un peu), vous verrez qu'il existe plein de codes graphiques. L'article wikipédia français les groupes sous le nom *codes-barres* (https://fr.wikipedia.org/wiki/Code-barre) (idem en anglais), ce qui est maladroit à mon avis, puisque ce que j'appelle (naturellement) codes-barres ne sont que les codes qui ont vraiment des barres, en 1D en général... Mais bref. Il y a donc plusieurs sortes de codes-barres :

- QR code (pour l'identification de pièces automobiles au Japon, à l'origine...) ;
- Datamatrix (classique aussi) ;
- CrontoSign (au hasard, il est assez original) ;
- etc.

Si vous observez un peu autour de vous (indices : colis & lettres, pièces électroniques...). Ils ont tous leurs caractéristiques, points forts et faibles, certains sont dans le domaine publique, d'autres sont brevetés, ou sous diverses licenses, etc.

Vous devez donc **inventer** un **nouveau** code.

"Un peu d'aide ?"

Pensez à vos **symboles** : il faut trouver l'équilibre entre la richesse (quantité d'information par symbole) et la facilité de lecture. Exemple : prenons les couleurs comme symboles. Vous avez des cellules de n'importe couleur RVB/RGB (Rouge/Vert/Bleu, 8 bits par composante, classique) vous avez :

- *256 * 256 * 256 = 16777216* **symboles possibles** ;
- *8 + 8 + 8 bits = 24 bits* par **symbole** (autre manière de voir) ;
- Mais est-ce que la caméra fera la différence entre *(255,255,255)* et *(255,254,255)* ? (indice : non).

Évitez donc les symboles trop riches, ici des couleurs trop précises.

La disposition peut être utile. Imaginons un exemple très simple : les symboles sont des lettres (==> une “cellule” = une lettre affichée). Je veux encoder “bonjour ca va ou” sur une matrice 4x4 :

Ordre “naturel” :

```
-------------
|1 |2 |3 |4 |
|5 |6 |7 |8 |
|9 |10|11|12|
|13|14|15|16|
-------------
```

on a donc

```
---------
|b|o|n|j|
|o|u|r| |
|c|a| |v|
|a| |o|u|
---------
```

Mais si on perd six caractères, en haut à gauche ?

```
---------
|#|#|#|j|
|#|#|#| |
|c|a| |v|
|a| |o|u|
---------
```

On obtient le message : “###j### ca va ou” : difficile à comprendre.

Le designer rusé change l’ordre (shuffle) :

```
-------------
|5 |9 |2 |12|
|11|14|7 |4 |
|15|1 |16|10|
|8 |13|3 |6 |
-------------
```

ce qui donne donc, si je ne me suis pas trompé

```
---------
|o|c|o|v|
| | |r|j|
|o|b|u|a|
| |a|n|u|
---------
```

Mais si on perd six caractères, en haut à gauche ?

```
_________
|#|#|#|v|
|#|#|#|j|
|o|b|u|a|
| |a|n|u|
_________
```

On obtient le message : “b#nj#u# #a#va#ou”. J’arrive mieux à lire : il n’y a pas de mot complet perdu.

En fait, j’ai *découplé* la position dans le message et la position dans l’affichage : en faisant ça j’augmente la robustesse face aux erreurs qui touchent des pixels voisins (la tache de café par exemple).

Dans mon code, il n’y a aucune redondance donc il y a une perte d’information (en plus, les lettres sont difficiles à reconnaître, surtout de loin). Le code ne corrige pas les erreurs tout seul, c’est mon cerveau qui corrige les erreurs parce qu’il sait parler et reconnaît le mot “bonjour” quand il voit “b.nj.u.” (mon cerveau a fait pas mal de mots fléchés). Dans le cas du QR code (et probablement d’à peu près tous les autres codes), les erreurs sont corrigées par le code lui-même, grâce à de la redondance introduite dans l’encodage. Je ne sais pas comment, je n’ai pas cherché.

Exemple le plus simple de redondance : vous mettez les cellules en double. Si vous en perdez une, il vous en reste une. Inconvénient : deux fois moins de place. Mieux : https://fr.wikipedia.org/wiki/Code_de_Hamming.

Il y a d’autres paramètres sur lesquels vous pouvez jouer ! Et les exemples précédents ne font qu’effleurer les problématiques impliquées, vous pouvez aller plus loin.

“Comment dessiner ?”

Pour des codes simples, vous pouvez dessiner en console, je pense (“#” fait un pixel noir, " " un pixel blanc par exemple).

Si vous voulez faire des images un peu plus évoluées, vous pouvez utiliser une bibliothèque graphique...

- “*Turtle graphics is a popular way for introducing programming to kids.*” https://docs.python.org/3.3/library/turtle.html?highlight=turtle
- Java et JavaFX...
- SFML (C++) ou SDL (C) avec des *sprites* qui sont vos *cellules*... ou directement au pixel, si vous voulez
- Javascript et canvas...
- SVG...

Comme vous voulez !