

# Rapport de projet - Détection et reconnaissance des lignes de métro

(tiré du sujet)



La société IMAGIK développe des lunettes équipées d'une caméra permettant à des personnes malvoyantes de se mouvoir dans le métro parisien grâce à des indications de direction. La caméra envoie les données à un centre de traitement qui se charge de convertir les images en indications pour l'utilisateur.

Nous sommes chargés de la détection des lignes de métro.

Pour le prototype de notre système, nous utilisons Matlab.

“Ainsi, le programme de démonstration demandé devra fournir pour chaque image la localisation des signes de lignes de métro détectés et le numéro de la ligne pour chaque signe.”

Notre module est donc chargé de convertir une image en un tableau indiquant les coordonnées de l'image où se trouve la pastille de la ligne de métro et son numéro.

A des fins d'évaluation, notre programme stockera les résultats dans une matrice BD de 6 colonnes, chaque ligne correspondant à une détection et chaque colonne à :

[Nom Image, x1, x2, y1, y2, Numéro de Ligne]

Ce problème s'inscrit dans deux domaines intervenant dans celui de la vision par ordinateur: le traitement d'images et l'analyse d'images.

La partie traitement consiste à modifier les pixels de l'image pour permettre à la partie chargée de l'analyse d'en tirer des informations utiles.



<b>Etat de l'art</b>	<b>3</b>
<b>Description des méthodes</b>	<b>5</b>
Etape 1: trouver les coordonnées des boîtes englobantes	5
Detection image originale	5
Detection image traitée	7
Saturation des blancs	7
Filtrage	8
Rassemblement des détections et coordonnées de boîtes englobantes	9
Diagramme de flux étape 1	10
Etape 2: Analyse des sous-images	11
Balance des blancs	11
Extraction d'images binaires de chiffres	11
Détermination des seuils optimaux	12
Suppression du fond	14
Rejet des petites régions	15
Séparation des chiffres	16
Similiarité	17
Détermination du nombre	18
Diagramme de flux étape 2	20
<b>Résultats expérimentaux et discussion</b>	<b>21</b>
Critique des méthodes utilisées, qualité de la détection	21
Imfindcircles	21
Corrélation	21
Critique du temps de traitement	21
Utilisation de l'IA	22
<b>Conclusions et perspectives</b>	<b>23</b>

# 1. Etat de l'art

La vision par ordinateur est un domaine multidisciplinaire qui traite de la manière dont un ordinateur peut extraire des concepts à partir d'une image.

La multidisciplinarité de ce domaine est en partie dû au fait qu'il touche l'acquisition, le traitement et l'analyse des images.

Les domaines dont la vision par ordinateur est à l'interface sont donc: l'intelligence artificielle, le traitement du signal, la physique, l'ingénierie de l'information, la neurobiologie, la géométrie et les mathématiques.

Il semble y avoir une tendance en augmentation depuis ces dernières années quant au fait que les progrès dans le domaine de la vision par ordinateur soit intimement liés à ceux de l'intelligence artificielle.

Classification d'images:

Question: Comment, à partir d'une image en déduire un label sachant que l'on possède un ensemble d'images similaires labellisées ?

Pour ce genre de problème, les chercheurs prennent chaque pixel des images labellisées et s'en servent pour entraîner un réseau de neurones d'apprentissage profond. Ce programme est ensuite capable en lisant les pixels de l'image à classifier de lui donner un label.<sup>1</sup>

La technologie la plus utilisée dans ce domaine est les réseaux de neurones à convolution (CNN). Particulièrement utiles en analyse d'image, ils permettent d'analyser l'image par blocs, puis par sous-blocs, et ainsi de suite. La grande force des CNNs réside dans le fait qu'ils soient capables de détecter des patterns dans l'entourage proche d'un pixel.

Contrairement au CNN, le perceptron multi couche, qui lui analyse l'image dans son ensemble est vulnérable aux translations des patterns dans l'image.<sup>2</sup>

Détection d'objets:

Question: Comment détecter un objet particulier dans une image ?

Exemple d'applications: Domaine médical, chaînes de productions...

Là encore, l'intelligence artificielle est au centre du processus. Pour résoudre ce genre de problème, on utilise des techniques à base de CNN: R-CNN, Fast R-CNN, Faster R-CNN. Ces modèles apportent chacun leurs lots d'améliorations, augmentant ainsi toujours plus la vitesse de détection. Le résultat obtenu est une boîte englobante affublée d'un label et d'un pourcentage de détection sur l'image source.

---

<sup>1</sup> "The 5 Computer Vision Techniques That Will Change How You See ...." 12 Apr. 2018, <https://heartbeat.fritz.ai/the-5-computer-vision-techniques-that-will-change-how-you-see-the-world-1ee19334354b>. Accessed 18 Jun. 2019.

<sup>2</sup> "What is the difference between convolutional neural networks and ...." 14 Sep. 2016, <https://stats.stackexchange.com/questions/234891/what-is-the-difference-between-convolutional-neural-networks-and-deep-learning>. Accessed 18 Jun. 2019.

Suivi d'objets:

Question: Comment suivre un objet dans une image ?

Exemple d'applications: Vidéo surveillance, voitures autonomes, robotique, réalité augmentée...

Deux méthodes n'incluant pas l'IA sont le suivi basé sur les kernels et le suivi de contours.<sup>3</sup>

La première consiste en la maximisation d'une mesure de similarité, la seconde consiste à ajuster le contour à chaque frame pour qu'il s'adapte à nouveau aux formes de l'objet.

Deux méthodes incluant l'IA étaient utilisées pour le suivi d'objet ces dernières années: les méthodes discriminantes et génératives. Au fil du temps la méthode discriminante prend peu à peu le pas sur son homologue, en effet, lorsqu'en présence d'une grande quantité de données, celle-ci montre de meilleures performances.<sup>4</sup>

Egomotion:

Question: Comment déterminer le mouvement dans l'espace de la caméra en fonction de sa séquence d'images ?

Exemple d'applications: Voitures autonomes, systèmes navigants...

Ce processus intervient dans la localisation et cartographie simultanée (SLAM), qui vise à être utilisé dans les voitures autonomes.

Estimation de pose:

Question: Comment estimer la position relatives d'objets dans une image ?

Exemple d'applications: Voitures autonomes, robotique...

On peut utiliser plusieurs approches : méthodes géométriques ou analytiques, algorithme génétique, IA.

La méthode géométrique consiste à effectuer des calculs dans les plans 2D et 3D avec une connaissance préalable de la morphologie de l'objet pour déterminer la pose. L'intelligence artificielle est quant à elle capable d'estimer la pose de l'objet après entraînement sur un set de données avec différentes poses.

Segmentation sémantique:

Question: Comment segmenter l'image en fonction de son contenu, pixel par pixel.

Exemple d'applications: Surveillance du trafic routier, voitures autonomes, imagerie satellite...

A cette fin, il existe deux méthodes : les FCN (fully convolutional network) et les encodeurs-décodeurs convolutionnels. Il semble que dans le monde de la recherche la première solution soit la plus utilisée pour la segmentation sémantique.

Segmentation d'instance:

Question: Comment segmenter une image en fonction des instances de ses objets, pixel par pixel ?

---

<sup>3</sup> "Video tracking - Wikipedia." [https://en.wikipedia.org/wiki/Video\\_tracking](https://en.wikipedia.org/wiki/Video_tracking). Accessed 18 Jun. 2019.

<sup>4</sup> "Explain to Me: Generative Classifiers VS Discriminative Classifiers." 8 Jan. 2014, <http://www.chioka.in/explain-to-me-generative-classifiers-vs-discriminative-classifiers/>. Accessed 18 Jun. 2019.

Utilisation d'un réseau Mask R-CNN. Celui-ci est composé d'un réseau Faster R-CNN pouvant être utilisé dans le cas de la détection d'objets avec une branche dédiée spécifiquement pour la partie "instances". Cette branche est composée d'un FCN et d'un masque binaire.

## **2. Description des méthodes**

Notre système effectue à plusieurs instants des étapes de traitements et d'analyse d'image. Nous nous sommes attachés durant ce projet à découper le code en fonctions afin de pouvoir le rendre plus lisible et le modifier ou le déboguer plus facilement.

Nous avons décidé de découper le problème en deux parties. La première partie consiste à récupérer des sous images contenant la pastille des lignes de métro. Avec cette étape, nous pouvons ainsi récupérer les coordonnées des boîtes englobantes des potentielles lignes détectées. La seconde consiste à effectuer des traitements et analyser la sous image afin d'en déduire la ligne qui s'y trouve. Cependant, avant traitement par la seconde partie, nous ne savons pas quel est le numéro de la ligne ni même si ce que nous avons détecté est bien une ligne de métro.

### **Etape 1: trouver les coordonnées des boîtes englobantes**

#### Detection image originale

Afin de récupérer les sous-images contenant les potentielles lignes de métro nous devons d'abord déterminer par un quelconque moyen les coordonnées de ces sous images. Pour cela, nous avons eu l'idée d'utiliser la fonction `imfindcircles` de matlab. Cette fonction prend en entrée certains paramètres dont une image où l'on souhaite trouver des cercles et renvoie en sortie une matrice contenant les positions de leurs centres et un vecteur contenant leurs rayons respectifs.



Exemple d'utilisation d'imfindcircles et visualisation des 5 détections les plus significatives<sup>5</sup>

Cette fonction qui exploite la transformée circulaire de Hough prend en entrée d'autres paramètres tels que les rayons des cercles à trouver, la sensibilité de la fonction de détection et le seuil pour la détection des gradients. Nous avons choisi de détecter les cercles de rayons entre 15 et 100, car à vue d'oeil cela semble être la plage de valeurs dans les photos fournies que ceux-ci peuvent prendre. Pour le choix de la sensibilité de la fonction de détection, celui-ci a aussi été fait à vue d'oeil et grâce au site Mathworks<sup>6</sup>. Nous avons choisi 0.92 car cette valeur nous permettait lors de nos tests de détection d'obtenir le plus d'information pertinentes possibles tout en minimisant les fausses détections (au delà de 0.93, le nombre de cercles que détecte la fonction augmente drastiquement).

L'application de cette fonction sur les images originales ne donnant pas toujours un résultat satisfaisant (certaines pastilles non-détectées), nous avons alors pensé à appliquer cette fonction sur des images traitées cette fois-ci, et à rassembler les résultats.

Nous avons donc essayé plusieurs traitements différents avant d'arriver à notre méthode actuelle. Celle-ci consiste à saturer les blancs, à passer l'image en gris, puis à appliquer un filtre passe-bas à l'image.

<sup>5</sup> "Find circles using circular Hough transform - MATLAB imfindcircles ...."  
<https://fr.mathworks.com/help/images/ref/imfindcircles.html>. Accessed 16 Jun. 2019.

<sup>6</sup> "Detect and Measure Circular Objects in an Image - MATLAB ...."  
<https://fr.mathworks.com/help/images/detect-and-measure-circular-objects-in-an-image.html>. Accessed 16 Jun. 2019.



## Detection image traitée

### *Saturation des blancs*

Ce choix de traitement a été motivé par le fait que d'après nos observations, la fonction *imfindcircles* repère mieux les objets circulaires contrastant plus fortement avec leur fond. Puisque dans tous les cas les pastilles de métro se trouvent sur fond blanc, nous avons alors décidé de saturer les pixels s'approchant le plus du blanc grâce à un seuil fixé en HSV tel que:

$$\left\{ p(x, y) = (255, 255, 255) \quad \text{if } (saturation(x, y) \leq 30) \cap (value(x, y) > 55) \right.$$

Pour éviter que certaines pastilles très claires (comme celles de la ligne 7) soient incluses dans cette formule et mises en blanc, nous avons exclu du traitement les pixels dont la couleur en approchait le rose.

## Filtrage

L'image résultante ne permettait cependant pas une détection satisfaisante, en effet, les pastilles contrastent mieux avec leur fond mais elles ont perdu pour certaines un peu de leur circularité, mettant à mal leur détection par la fonction. C'est pour palier ce problème que nous avons choisi d'appliquer un filtre gaussien passe-bas léger (pixel de coupure  $r_0 = 100$ ), afin "d'arrondir" les cercles. Le code de la fonction de filtrage `glp.m` a été en grande partie tirée du site de MathWorks FileExchange et écrite par Mohamed Athiq<sup>7</sup>.

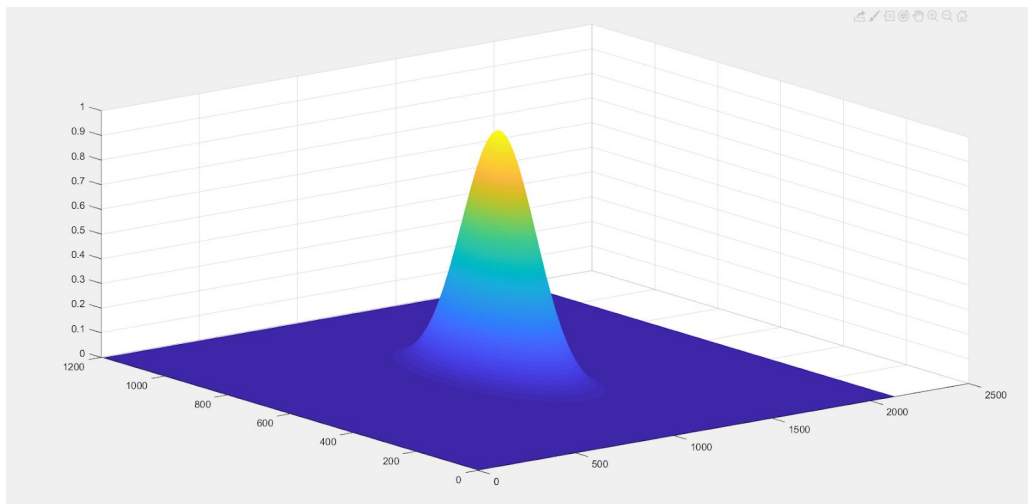
La fonction originale prenait en entrée une fft d'image en niveau de gris mais nous l'avons modifiée pour qu'elle prenne une image en rgb et retourne en sortie l'image filtrée en gris.

L'opération effectuée dans le domaine fréquentiel par cette fonction peut être modélisée mathématiquement comme suit:

$$H(u, v) = e^{-r^2/(2r_0)^2}, r = \sqrt{(u - (\dim X/2))^2 + (v - (\dim Y/2))^2}$$

Avec  $r_0$  le pixel de rayon de coupure indiqué en paramètre de la fonction.

Voici le filtre représenté dans le domaine fréquentiel en 3 dimensions:



Une fois cette opération réalisée, et après application de la fonction `imfindcircles` à l'image traitée, nous obtenons une seconde matrice de centres et un second vecteur de rayons. Nous avons pensé à privilégier une méthode plutôt qu'une autre, mais finalement, sur certaines images les détections issues de l'image originale et traitée se complètent. Nous faisons donc passer les résultats de nos deux détections (2 matrices de centres et 2 vecteurs de rayons) dans une fonction chargée de les fusionner.

---

<sup>7</sup> "Frequency domain filtering for grayscale images - File Exchange ...."

<https://de.mathworks.com/matlabcentral/fileexchange/40579-frequency-domain-filtering-for-grayscale-images>. Accessed 16 Jun. 2019.



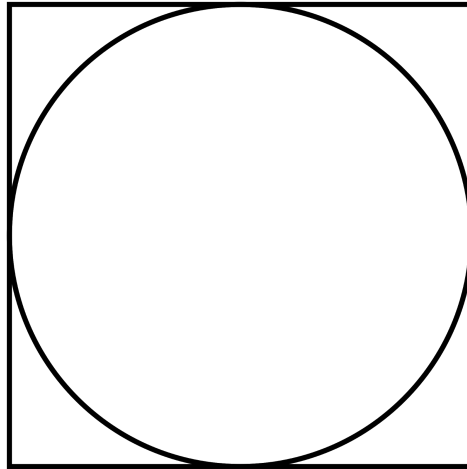
## Rassemblement des détections et coordonnées de boîtes englobantes

C'est la fonction `mergeCenters` qui a ce rôle et également celui de se débarrasser des doublons, c'est à dire les rayons et centres détectés à la fois par la première fonction et par la deuxième. Pour détecter les doublons, la fonction vérifie si, dans la première matrice et le premier vecteur, pour une même détection (rayon et centre), celle ci est aussi présente dans la seconde matrice et le second vecteur.

La similarité des détections est déterminée comme suit:

$$(Rayon2(j) - 15 \leq Rayon1(i) \leq Rayon2(j) + 15) \cap (Centre2(j) - 10 \leq Centre1(i) \leq Centre2(j) + 10)$$

A partir des rayons et des coordonnées des centres, nous pouvons aisément obtenir les coordonnées d'une boîte englobante carrée.



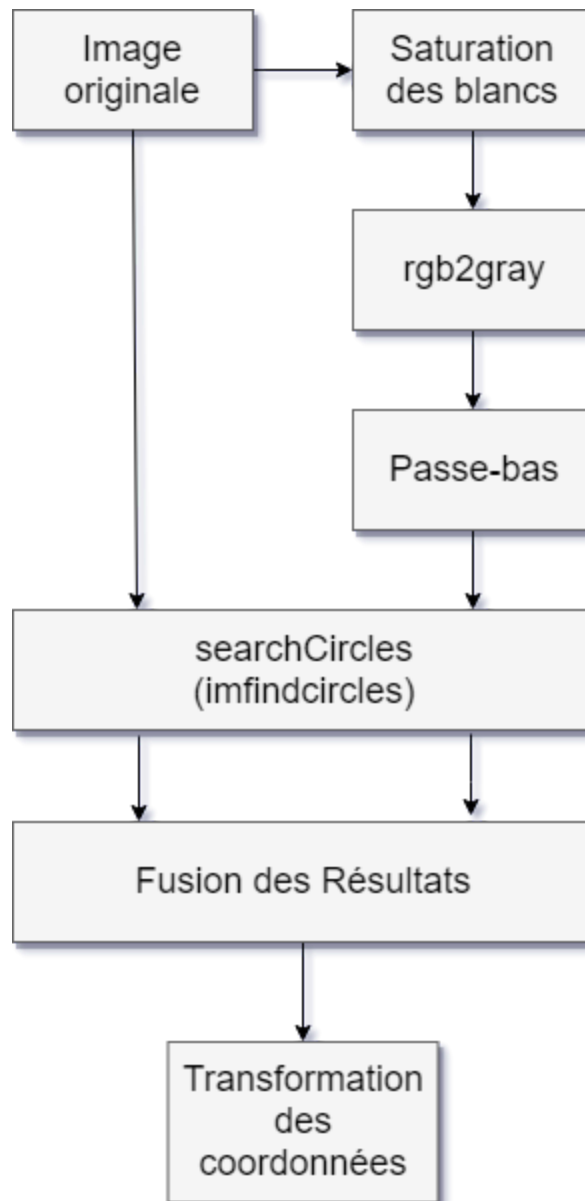
C'est ce qu'effectue la fonction `centersToCoords`, qui prend en entrée des centres et rayons et renvoie une matrice de coordonnées `[x1, x2, y1, y2]`.

Cette opération se présente mathématiquement de la manière suivante :

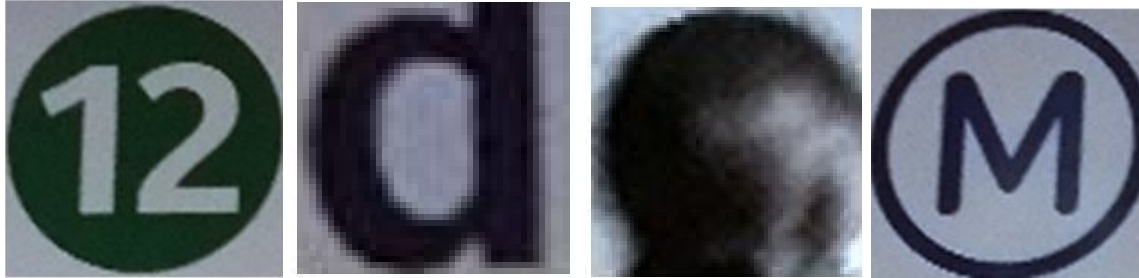
$$\begin{bmatrix} x1 \\ x2 \\ y1 \\ y2 \end{bmatrix} = \begin{bmatrix} xCenter \\ xCenter \\ yCenter \\ yCenter \end{bmatrix} + \begin{bmatrix} -radius \\ radius \\ -radius \\ radius \end{bmatrix}$$

## Diagramme de flux étape 1

Pour récapituler, voici un diagramme de flux de notre système pour la partie de détermination des boîtes englobantes.



## Etape 2: Analyse des sous-images



Une fois les sous images acquises, le programme va passer à l'étape suivante, c'est à dire extraire la ligne (ou l'absence de ligne) correspondante.

Notre approche a d'abord consisté à effectuer une corrélation entre les sous images obtenues et les échantillons de ligne de métro fournis. Cette dernière était effectuée avec les images rgb et peu efficace. Nous avons alors décidé d'essayer de traiter les sous images afin de pouvoir y effectuer une corrélation binaire. Cela a très bien fonctionné et c'est finalement cette approche que nous avons retenu.

### Balance des blancs

Pour commencer nous redressons la teinte des blancs de la sous-image car nous aurons plus tard besoin d'établir des clusters de couleurs dans celle-ci. Pour cela nous utilisons les fonctions matlab `illumgray` puis `chromadapt`.

### Extraction d'images binaires de chiffres

L'étape suivante consiste à extraire des sous images en binaire des potentiels chiffres se trouvant dans la sous-image. Souhaitant effectuer plus tard une corrélation sur ceux-ci avec les images binaires fournies dans PICTO, il faut que si un nombre est effectivement extrait il ressemble le plus possible à ces dernières afin de maximiser le résultat de la comparaison.



Notre fonction `extractNumbers` prend en entrée une sous-image et renvoie en sortie une cellule de matrices binaires, chaque matrice devant correspondre à un potentiel nombre.

Dans ces matrices, on aimerait que les pixels formant le nombre soient égales à 1, le reste à zéro. Pour cela on utilise un/des seuils déterminés par la méthode d'Otsu sur les sous images.

### *Détermination des seuils optimaux*

La méthode d'Otsu nécessite de spécifier le nombre de clusters à trouver dans la sous image afin d'en déduire les seuils minimisant la variance intra-classe. Cette méthode peut être modélisée mathématiquement par les formules suivantes:

$$\sigma_B^2 = \sum_{k=0}^{K-1} \omega_k (\mu_k - \mu_T)^2$$

Avec K le nombre de clusters que l'on souhaite former,  
 $\omega_k$  la probabilité qu'un pixel tiré au hasard appartienne à la classe k,  
 $\mu_k$  le niveau de gris moyen de la classe k,  
 $\mu_T$  le niveau de gris moyen sur l'ensemble de l'image,

On cherche:

$$\{k_1, \dots, k_{K-1}\} = \operatorname{argmax}(\sigma_B^2(k_1, \dots, k_{K-1}))$$

On doit donc spécifier le nombre de clusters à trouver dans l'image. Or on remarque que les sous-images peuvent être découpées parfois en 2, parfois en 3 classes comme le montre les images ci-dessous:



En effet, les sous images étant carrées, on obtient dans tous les cas la couleur blanche au niveau des coins, celle de la pastille et celle du chiffre au centre. On remarque que c'est lorsque le chiffre au centre est noir qu'on peut découper l'image en 3 classes, sinon, en deux. Il nous a donc fallu trouver une méthode pour déterminer le nombre de seuils à spécifier dans la fonction `multithresh` de matlab, qui utilise la méthode d'Otsu précédemment décrite. Cette fonction prend en entrée l'image en niveau de gris à seuiller et le nombre de seuils à trouver. Elle renvoie en sortie une matrice de taille égale à celle de l'image, faisant correspondre pour chacun de ses pixels la classe de ceux de l'image source.

Notre programme détermine donc le nombre optimal de clusters de la manière suivante:

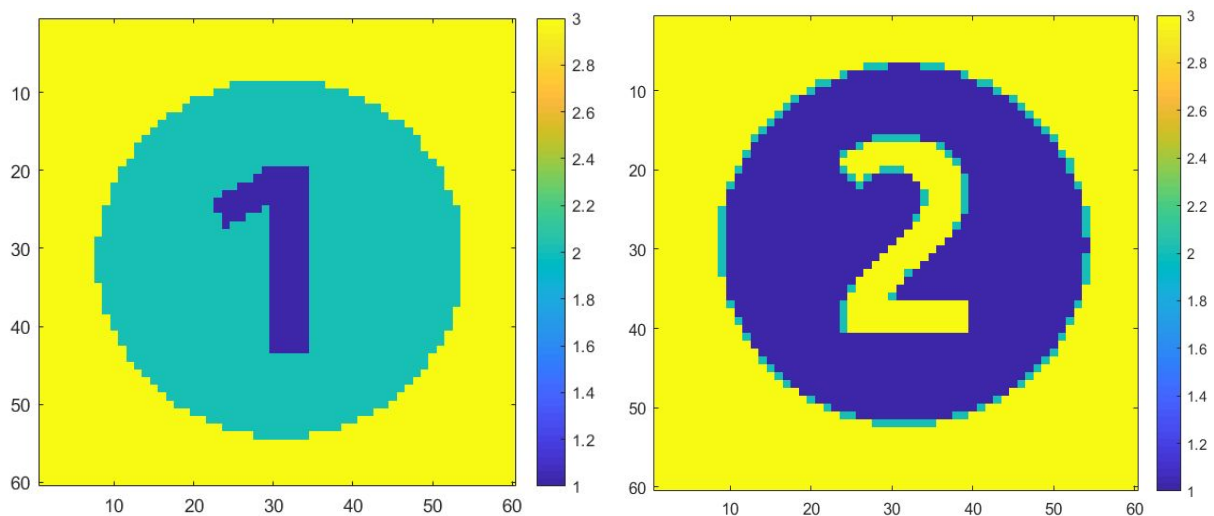
Les classes étant triées par intensité moyenne de pixels croissante,

Si la seconde classe issue du résultat du seuillage pour 3 classes est la plus petite, alors le nombre des classes optimal est 2, sinon 3.

En effet, si l'on est dans le cas d'une image avec 3 classes (exemple: blanc, jaune, noir pour la ligne 1), chaque couleur sera attribuée à un cluster dans cet ordre : 1 = noir, 2 = jaune, 3 = blanc, ou en généralisant: 1 = noir, 2 = couleur de la pastille, 3 = blanc.

Dans ce cas ci, la classe correspondant à la pastille n'est jamais la moins peuplée.

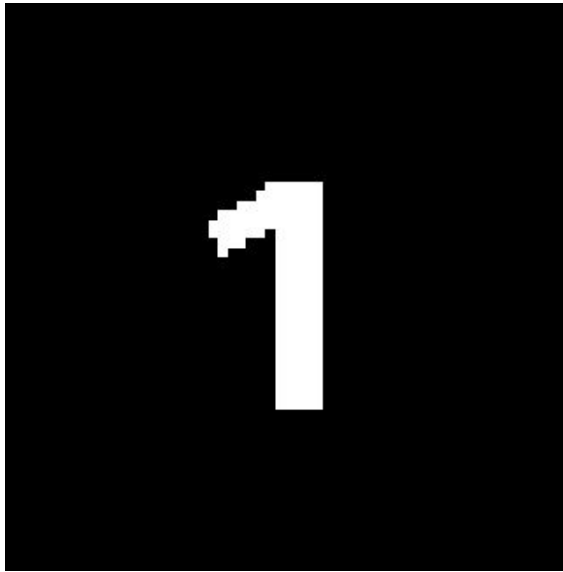
Tandis que dans le cas d'une image avec 2 classes (exemple: blanc et bleu pour la ligne 2), le programme va classer une partie des pixels blancs et de la couleur de la pastille dans deux classes distinctes, mais va forcément générer une 3ème classe avec les pixels à l'interface entre la pastille et le blanc. Cette 3ème classe est en seconde position car composée d'un mélange entre les pixels les plus sombres (pastille) et les plus clairs (blanc) et est également toujours la moins peuplée.



A gauche: 3 couleurs, 3 clusters A droite: 2 couleurs, 3 clusters

Après avoir déterminé le nombre de clusters optimaux, le programme crée une image binaire avec à 1 les pixels correspondant à : la classe la plus sombre si clusters = 3, la plus claire si clusters = 2, ce qui nous permet dans les deux cas de récupérer au moins les pixels de la classe correspondant au nombre central.

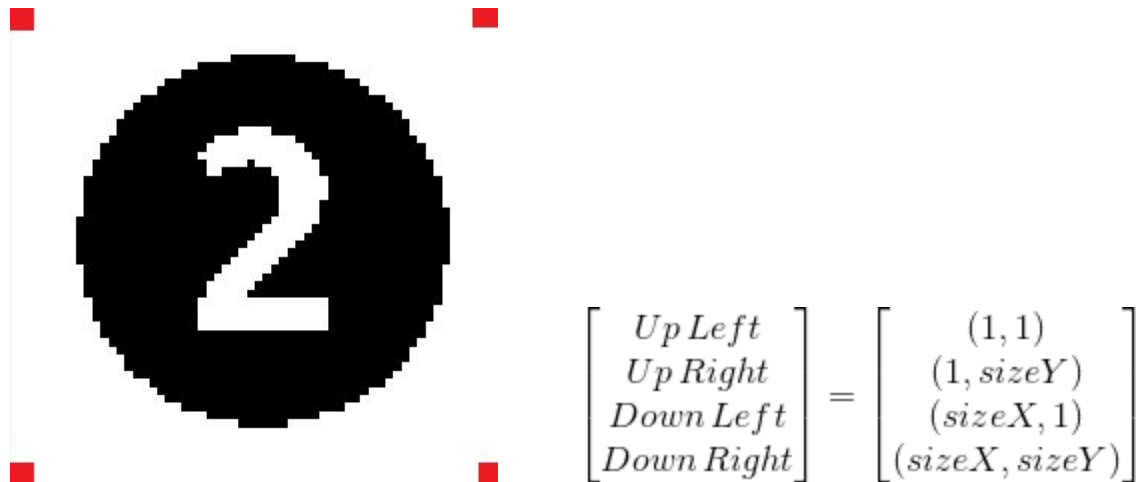
### *Suppression du fond*



A gauche: seul le chiffre est à 1. A droite: le chiffre et l'arrière plan sont à 1

Comme nous prévoyons d'utiliser la fonction `bwlabel` de matlab pour classifier les pixels de chaque chiffre détecté sur l'image pour exploitation future, nous devons d'abord supprimer le fond blanc des images bicolores. En effet, la classe des pixels du chiffre est la même que la classe des pixels du fond sur ces dernières.

Le programme effectue alors une détection binaire de régions contiguës grâce à la fonction `matlab grayconnected`. Celle ci est effectuée à 4 reprises, plaçant à chaque itération les graines aux niveau des coins de l'image binaire.

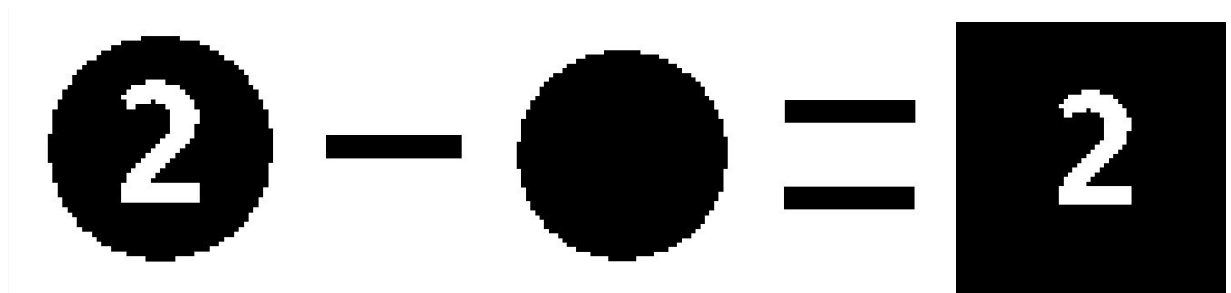


Les résultats sont stockés dans des matrices  $BW_n$  ( $n=1,...,4$ ).

Le programme réalise ensuite l'union des quatre régions:

$$BW = BW1 \cup BW2 \cup BW3 \cup BW4$$

Puis soustrait ces régions à l'image binaire de la ligne de métro:



Après avoir obtenu un chiffre en binaire exploitable, le programme utilise la fonction `bwlabel` pour classifier chaque région continue blanche.

### *Rejet des petites régions*

Le programme itère ensuite sur tous ces labels. Toutes les régions dont le nombre de pixels ne dépassent pas 5% du nombre de pixels totaux de l'image sont mis à zéro. Cela nous permet de supprimer les signaux indésirables se mêlant au nombre de la ligne.

A partir de ce moment, si la sous-image originale est celle d'une ligne de métro il ne nous reste normalement plus que des chiffres sur celle-ci. Si la sous-image était à la base une fausse

détection, le programme s'occupera de l'écarter plus tard. A ce stade, faux et vrais positifs de sous-images sont donc toujours mêlés.

### *Séparation des chiffres*

Afin de passer d'une sous image binaire entière à une sous image contenant un seul chiffre et dont l'apparence se rapproche des chiffres en noir et blanc trouvés dans le dossier PICTO, le programme va délimiter des boîtes englobantes pour chaque chiffre et ranger les matrices binaires résultantes dans une cellule.

Exemple :



Sous-image binaire de la ligne 12



Résultats du découpage des chiffres de la sous-image



## Similiarité

Notre fonction `templateMatching` prend en entrée une cellule et retourne une matrice de similarité contenant un indicateur de similarité compris entre 0 et 1, pour chaque chiffre et pour chaque fichier noir et blanc PICTO. Cet indice de similarité est déterminé en comparant chaque pixel binaire de l'image en entrée avec chaque pixel binaire des différentes images de PICTO redimensionnées pour coller aux dimensions de l'image d'entrée. Si les deux pixels sont de valeur égale, alors le score augmente de 1. A la fin de la comparaison entre les deux images, le score est divisé par le nombre total de pixels dans l'image afin d'obtenir un chiffre entre 0 et 1. Ce chiffre correspond donc à la proportion de pixels de coordonnées (x,y) dans l'image d'entrée égaux aux pixels de coordonnées (x,y) dans l'image PICTO. Cela correspond à effectuer l'opération booléenne XNOR entre les deux matrices, additionner tous les 1 qui s'y trouvent puis les diviser par le produit des dimensions des matrices.

Voici comment nous formulons le problème mathématiquement:

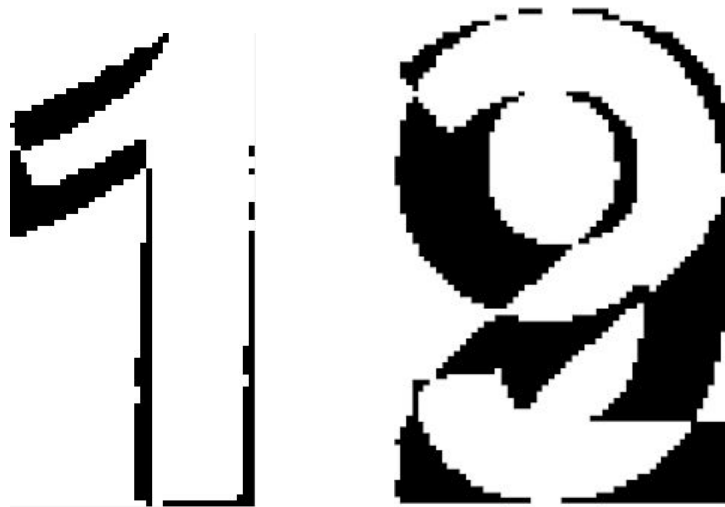
$$Resultat = XNOR(Input, Picto) = (Input \cap Picto) \cup (\overline{Input} \cap \overline{Picto})$$

$$Similarite = \sum_x^{\text{sizeX}} \sum_y^{\text{sizeY}} Resultat(x, y)$$

$$\text{Avec : } Input = \begin{bmatrix} Input(1, 1) & \dots & Input(1, \text{sizeY}) \\ \dots & \dots & \dots \\ Input(\text{sizeX}, 1) & \dots & Input(\text{sizeX}, \text{sizeY}) \end{bmatrix}$$

$$(\text{sizeX}, \text{sizeY}) = \text{size}(Input) = \text{size}(Picto) = \text{size}(Resultat)$$

Voici des exemples de matrices Résultat:



Gauche : Similarité entre un 1 détecté et le 1 de picto. Droite : Similarité entre un 2 et un 9.  
Plus il y a de blanc, plus d'indicateur de similarité sera élevé.

En général, l'indicateur de similarité est à 0.8 quand le bon nombre est détecté.


## Détermination du nombre

La détermination définitive du nombre se fait à partir de la matrice de similarité générée par la fonction précédente, `templateMatching`. La matrice de similarité est de dimension  $10 \times n$ . Le nombre  $n$  de colonnes dépend du nombre de chiffres potentiels extraits en binaire précédemment. Le nombre de lignes dépend du nombre d'images en noir et blanc fournies dans `picto` (représentant en noir et blanc les chiffres de 0 à 9) contre lesquels les chiffres extraits ont été analysés. La case (6,2) de la matrice correspond donc à l'index de similarité entre le second chiffre potentiel extrait et le pictogramme correspondant au chiffre 5.

La fonction `simToNum` prend en entrée la matrice de similarité et en déduit le numéro de la ligne de métro qui y est associée. Pour cela, elle calcule l'index de similarité le plus grand dans chaque colonne, en déduit le chiffre associé, le transforme en string, le concatène avec celui de l'autre colonne s'il y a lieu, puis le convertit en nombre. Ainsi 1 et 2 passent par le traitement `[1,2] => ['1', '2'] => '12' => 12`.

C'est lors du passage dans cette fonction que sont triés les faux positifs, en effet, si l'index de similarité maximum dans une colonne n'excède pas 0.78, alors la colonne entière n'est pas prise en compte.

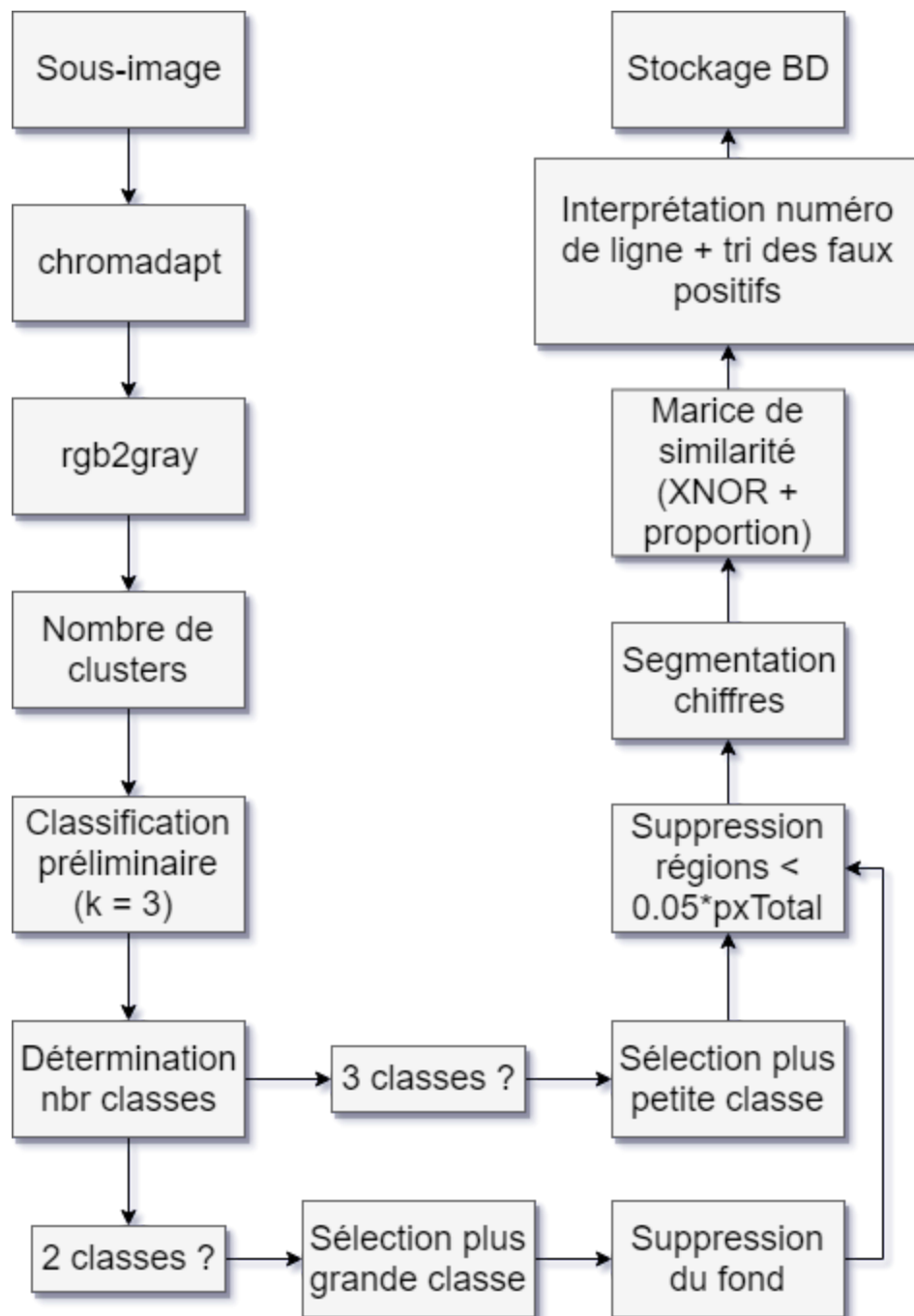
A la fin du traitement, la fonction retourne une cellule contenant le numéro de la ligne et l'index de similarité maximum de la matrice de similarité. Si rien n'a été détecté, la fonction retourne deux 0.

RESULTS <span>✕</span>		
 2x1 <a href="#">cell</a>		
	1	
1	12	
2	0.8406	
3		

Pour finir, nous écrivons les résultats dans une nouvelle ligne de la matrice BD.

## Diagramme de flux étape 2

Voici un diagramme de flux résumant les opérations effectuées par le système durant la seconde étape :



### **3. Résultats expérimentaux et discussion**

#### Critique des méthodes utilisées, qualité de la détection

##### Imfindcircles

La raison pour laquelle nos résultats ne sont pas parfaits est multiforme. L'une d'entre elles est l'utilisation d'imfindcircles. Parfois la fonction ne détectait pas certaines pastilles, comme c'est le cas avec l'image 5, ce qui nous a motivé à superposer les détections avec une image traitée.

Un des problèmes que l'utilisation de la fonction génère est le nombre élevé de fausses détections au delà d'un certain seuil de sensibilité. La fonction est néanmoins performante lorsqu'il s'agit de trouver des cercles bien contrastés avec leur fond.

Dans une application en situation réelle, la fonction imfindcircle pourrait poser un problème si la pastille est vue en perspective. En effet, la projection de celle ci sur la caméra lui ferait perdre sa forme ronde et compromettrait sa détection par la fonction.

De plus, de part le nombre important de détections générés par celle ci, nous pourrions également saturer centre de traitement dans le cloud. Enfin, il suffirait d'un simple dessin d'un chiffre à l'intérieur d'un cercle (comme par exemple avec une pub pour le loto), pour créer un faux positif passant sans encombre les étapes de traitement de notre module, puisqu'à aucun moment nous n'utilisons les couleurs des pastilles pour les identifier.

##### Corrélation

La détection des nombres après segmentation est également sujette aux problèmes. Puisque nous effectuons un template matching, un chiffre en rotation de quelques degrés pourrait perturber la détection et donner des résultats non-significatifs pour la détection. Il suffit donc que la personne porte ses lunettes un peu inclinées ou que sa tête le soit, et elle pourrait bien passer à côté d'une indication de station de métro sans s'en apercevoir.

#### Critique du temps de traitement

Comme vous aurez pu le constater si vous avez lancé notre programme, le temps de traitement est plutôt long par image. Nous n'avons pas connaissance de la rapidité de traitement des clouds, mais l'on peut supposer qu'avec plusieurs utilisateurs et un framerate assez haut, les

centres de traitement tourneraient à plein régime et les performances du système s'en trouveraient réduites. Les principales fonctions ralentissant le temps de traitement sont la fonction de filtrage dans le domaine spectral et la fonction imfindcircles.

## Utilisation de l'IA

Notre programme est un programme de vision par ordinateur utilisant des techniques de traitement d'images coûteuses en puissance et temps de traitement. Avec l'intelligence artificielle, nous pourrions réduire le temps de traitement de manière significative. Nous pourrions en particulier utiliser des techniques de type Faster R-CNN avec en données d'entraînement les sous-images dont les coordonnées se trouvent dans les fichiers excels fournis.

## 4. Conclusions et perspectives

Les méthodes que nous avons utilisé pour ce programme nous permettent de détecter et reconnaître les stations affichées dans la plupart des cas. Cependant, cette solution ne serait pas applicable en situation réelle car le temps de traitement est trop long et la reconnaissance de chiffres est vulnérable à la rotation de l'image. En effet, nous avons utilisé une corrélation entre images binaires pour reconnaître les numéros et la fonction `imfindcircles` pour localiser les pastilles. L'utilisation notamment d'un filtre passe-bas sur chaque image et de l'utilisation de la fonction `imfindcircles` à deux reprises ont grandement participé à la relative lenteur de notre programme.

L'un des axes d'amélioration sans IA que nous pourrions considérer est d'identifier la position des pastilles au départ en analysant les couleurs présentes dans l'image. Cependant, ayant étudié cette approche auparavant et échoué, nous nous sommes finalement dirigés vers l'utilisation de la transformé de Hough.

L'axe d'amélioration intégrant l'IA pourrait consister à entraîner un réseau de neurones de détection d'objet de type Faster R-CNN. Nous pourrions ainsi potentiellement atteindre un score de détection plus haut en moins de temps de traitement.

Grâce à la réalisation de ce programme nous avons pu avoir une meilleure perceptions des enjeux liés au prototypage d'un programme de reconnaissance d'objets. Nous avons pu parfaire nos connaissances sur Matlab, en vision par ordinateur et avoir un début de vision d'ensemble des techniques utilisées à l'heure actuelles.

Nous remarquons que l'intelligence artificielle prend beaucoup de place dans le domaine, une tendance qui a l'air d'être la même dans les autres disciplines scientifiques, techniques et dans l'industrie.