UNIVERSITÉ
CAEN
NORMANDIE

Rapport du projet annuel

Analyseur de marqueurs visuels

Equipe :

- KHARROUBI Youcef

Encadrant :

- Mr CHAHIR Youssef

Examinateur :

- Mr COURTIEL Julien

Année 2021/2022

Sommaire :

1. Introduction

Ce projet est centré sur les symboles OILU, un nouveau système de numérotation basé sur des lignes. Ce dernier permet une meilleure concordance avec les valeurs des chiffres décimaux, et ouvre de nouvelles opportunités, qui ne sont pas possibles avec le système de numération décimale classique [1]. Ces nouveaux symboles sont utilisés dans le développement d'un nouveau type de marqueurs visuels qui ont de nombreuses applications potentielles: réalité augmentée ainsi que la navigation des drones.

2. Présentation du projet

Dans ce projet, nous explorons les symboles OILU ainsi que les marqueurs visuels produits à partir de ces symboles. Pour ce but, nous essayons de réaliser une application pour Android qui a pour fonctions les tâches suivantes:

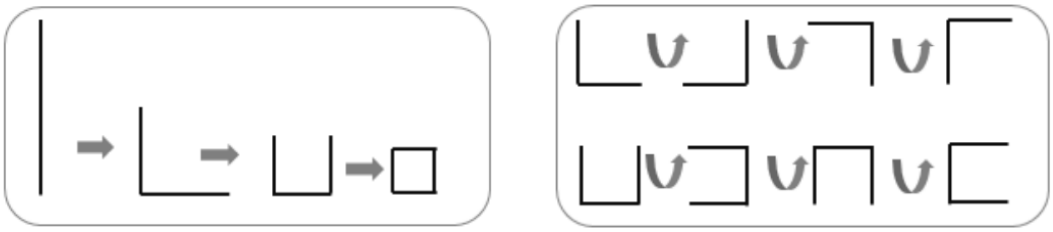
- Détection des marqueurs visuels;
- Identification des marqueurs visuels;
- Mise en forme 3D des marqueurs visuels avec la réalité augmentée.

3. Aperçu sur les symboles et les marqueurs visuels OILU

L'idée de départ pour pouvoir obtenir les symboles OILU qui représentent l'ensemble des chiffres décimaux est de plier un segment pour obtenir des rectangles à un, deux, trois ou quatre côtés qu'on pourra utiliser comme un système de numérotation (Figure 1.a). Ce système de numérotation sera donc composé de dix symboles:

- les symboles basiques qui sont les quatre rectangles obtenus après pliage d'un segment { □, |, ㄴ, ㅂ } et qui représenteront respectivement { 0, 1, 2, 3 } [1]
- Par rotations antihoraires successives de 90° des deux symboles { ㄴ, ㅂ} on obtient six autres symboles (Figure 1.b) qui serviront à représenter les six chiffres restants comme suite [1]:
    - ㄴ et ses différentes rotations représentent les chiffres pairs.

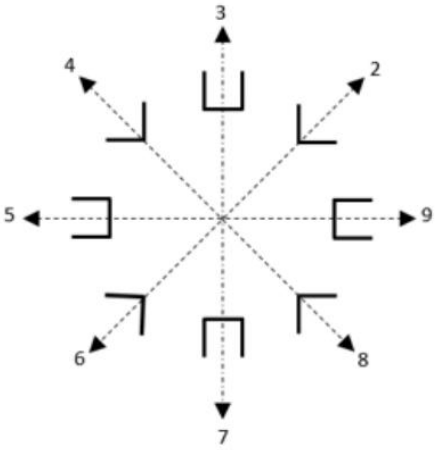
- ⊔ et ses différentes rotations représentent les chiffres impairs.



(a) Modèle de base (b) Génération des symboles restant

Figure 1: génération des symboles [1]

On obtient donc dix symboles qui nous serviront à représenter les chiffres décimaux. En excluant les symboles { □ et | } qui ne sont pas concernés par la rotation et sont automatiquement affectés aux chiffres 0 et 1, le reste des symboles est affecté (selon leurs orientations) aux 8 directions correspondantes (Figure 2.a) [1].



(a) Orientation des symboles

Digit	Affected Symbol
0	□
1	|
2	⌞
3	⊔
4	⌟
5	⊐
6	⌝
7	⊓
8	⌜
9	⊏

(b) Affectation des symboles

Figure 2: Affectation et codification des symboles [1]

Le tableau suivant (Figure 2.b) présente la correspondance entre les symboles générés et les valeurs des chiffres décimaux [1].

Sur le plan de la consommation énergétique, la symbolique OILU permet une réduction d'au moins 50% de l'énergie d'affichage : 25 segments contre 49 (pour les afficheurs classiques) (Figure 3.a) [1]. Le principal intérêt

de ce système, c'est qu'il permet de superposer des symboles sous forme pyramidale sans perdre la valeur des nombres construits. La figure 3.b présente un exemple de représentation pyramidale du nombre décimal (3172). Le sens de la lecture va de l'extérieur vers l'intérieur. Contrairement à la symbolique décimale numérique classique, la symbolique OILU permet de voir un nombre comme un objet avec ses quatre facettes. En effet, on peut extraire de chaque point de vue une valeur numérique différente. Ainsi, un groupe de nombres liés est formé. Dans le cas de notre exemple, les nombres associés sont : 3172, 9158, 7136 et 5194. Toute valeur de facette permet de déduire les valeurs des autres facettes [1].

(a) Affichage serial (b) Affichage pyramidal des chiffres

Figure 3: Affichage des nombres et symboles OILU [1]

4. Détection des marqueurs

Dans ce projet, on commence d'abord par détecter les marqueurs visuels. On a, pour ce projet, résumé la détection d'un marqueur OILU à la détection de rectangle dans l'image donnée. Pour ce faire, plusieurs approches ont été explorées. Ce qui suit énumère les différentes approches étudiées lors de la réalisation de ce projet.

a. Première approche

Dans cette première approche, on explore les deux opérations de base de la morphologie mathématique qui sont la dilatation et l'érosion ainsi que la transformée de Hough et comment pouvons nous appliquer toutes ces techniques pour appréhender notre problème de détection de rectangle.

i. Morphologie mathématique

La morphologie mathématique est un ensemble d'opérations de traitement d'image qui permettent de nettoyer les images bruitées. La morphologie mathématique fait une transformation géométrique d'une image binaire, le traitement de la forme des objets et la suppression du bruit.

Pour les opérations de base, qu'on évoquera par la suite, la morphologie mathématique fait intervenir un élément structurant qui fait fonction d'un masque de convolution appliqué à une image binaire. Il existe plusieurs types d'éléments structurants de taille et de forme variées (carré, rectangle, ellipse, croix … etc.). Chacun de ces éléments structurants peuvent être utilisés en fonction de la complexité de l'image, la vitesse du traitement et l'objectif attendu de l'opération entreprise.

1. Opérations de base

Il existe deux types d'opérations morphologiques de base: l'érosion et la dilatation. Ces opérations de base sont duales, c'est-à-dire qu'elles se complètent.

a. Dilatation

L'effet de base de la dilatation sur une image binaire est d'agrandir progressivement les limites des régions de pixels de premier plan (c'est-à-dire les pixels blancs, généralement). Ainsi, les zones de pixels de premier plan augmentent en taille tandis que les trous dans ces régions deviennent plus petits.

La définition mathématique de la dilatation pour les images binaires est la suivante :

- Supposons que X est l'ensemble des coordonnées euclidiennes correspondant à l'image binaire d'entrée, et que K est l'ensemble des coordonnées de l'élément structurant.
- Soit K_x la translation de K telle que son origine soit en x.
- Alors la dilatation de X par K est simplement l'ensemble de tous les points tels x que l'intersection de K_x avec X est non vide.

On aura donc :

$$\epsilon_K(X) = X \oplus B = \{x \in image \,|K_x \cap X \neq \emptyset\}$$

Pour calculer la dilatation d'une image binaire d'entrée par cet élément structurant, nous considérons tour à tour chacun des pixels d'arrière-plan de l'image d'entrée. Pour chaque pixel d'arrière-plan (que nous appellerons pixel d'entrée), nous superposons l'élément structurant au-dessus de l'image d'entrée afin que l'origine de l'élément structurant coïncide avec la position du pixel d'entrée. Si au moins un pixel dans l'élément structurant coïncide avec un pixel de premier plan dans l'image en dessous, alors le pixel d'entrée est défini sur la valeur de premier plan. Si tous les pixels correspondants dans l'image sont en arrière-plan, cependant, le pixel d'entrée est laissé à la valeur d'arrière-plan.

b. Erosion

L'effet de base de l'érosion sur une image binaire est d'éroder les limites des régions de pixels de premier plan (c'est-à-dire les pixels blancs, généralement). Ainsi, les zones de pixels de premier plan diminuent de taille et les trous à l'intérieur de ces zones deviennent plus grands.

La définition mathématique de l'érosion pour les images binaires est la suivante :

- Supposons que X est l'ensemble des coordonnées euclidiennes correspondant à l'image binaire d'entrée, et que K est l'ensemble des coordonnées de l'élément structurant.
- Soit K_x la translation de K telle que son origine soit en x.
- Alors l'érosion de X par K est simplement l'ensemble de tous les points x tels que K_x est un sous-ensemble de x.

On aura donc :

$$\delta_K(X) = X \ominus B = \{x \in image \,|K_x \cap X \neq \emptyset\}$$

Pour calculer l'érosion d'une image binaire d'entrée par cet élément structurant, on considère tour à tour chacun des pixels de premier plan dans l'image d'entrée. Pour chaque pixel de premier plan (que nous appellerons pixel d'entrée), nous superposons l'élément structurant au-dessus de l'image d'entrée afin que l'origine de l'élément structurant coïncide avec les

coordonnées du pixel d'entrée. Si pour chaque pixel de l'élément structurant, le pixel correspondant dans l'image en dessous est un pixel de premier plan, alors le pixel d'entrée est laissé tel quel. Cependant, si l'un des pixels correspondants de l'image est en arrière-plan, le pixel d'entrée est également défini sur la valeur d'arrière-plan.

2. Application

Pour ce qui est de la détection du marqueur visuel OILU, on a choisi l'exemple illustré dans la Figure 4 pour élaborer cette première approche:

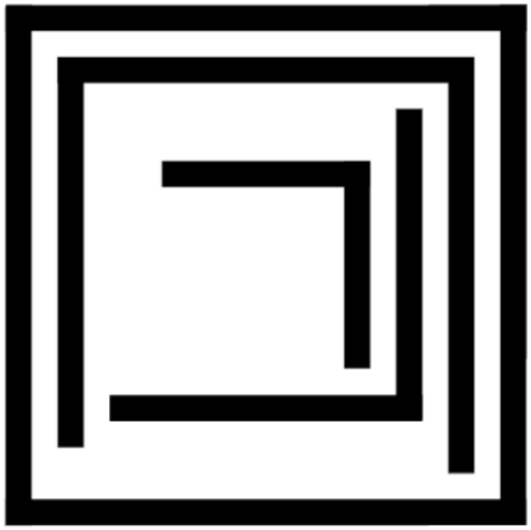
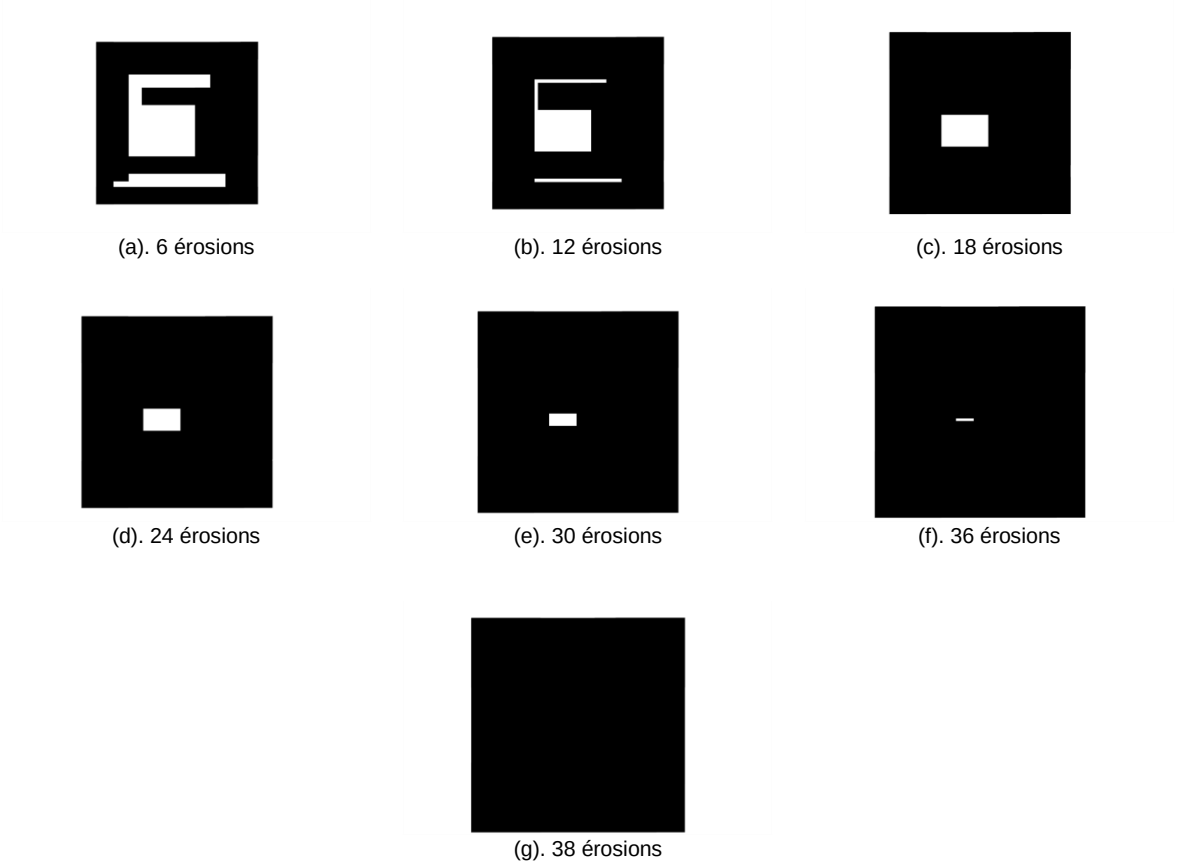


Figure 4: Exemple d'un marqueur visuel OILU.

Pour réaliser la détection du rectangle, l'idée de cette première approche est la suivante:

1. On voudra enlever le bruit qui est dans ce cas les pixels blancs se trouvant à l'intérieur du rectangle pour qu'à la suite on aura un rectangle de couleur uniforme qui sera noir. Cela sera fait grâce à des érosions successives jusqu'à en obtenir aucun pixel blanc à l'intérieur du rectangle. La Figure 5 montre l'évolution de cette étape à différents stades d'érosion.



(a). 6 érosions (b). 12 érosions (c). 18 érosions

(d). 24 érosions (e). 30 érosions (f). 36 érosions

(g). 38 érosions

Figure 5: Etats du marqueur visuel à différents stades des érosions successives.

2. Le résultat de la première étape nous donnent effectivement un rectangle de couleur uniforme noir (pas de pixels blancs), mais on remarque que la taille du rectangle n'est pas la même comparée à celle du marqueur visuel. Afin de rétablir la même taille, on peut effectuer le même nombre de dilatations successives que celui des érosions. Les résultats de cette technique sont illustrés dans la Figure 6 suivante:

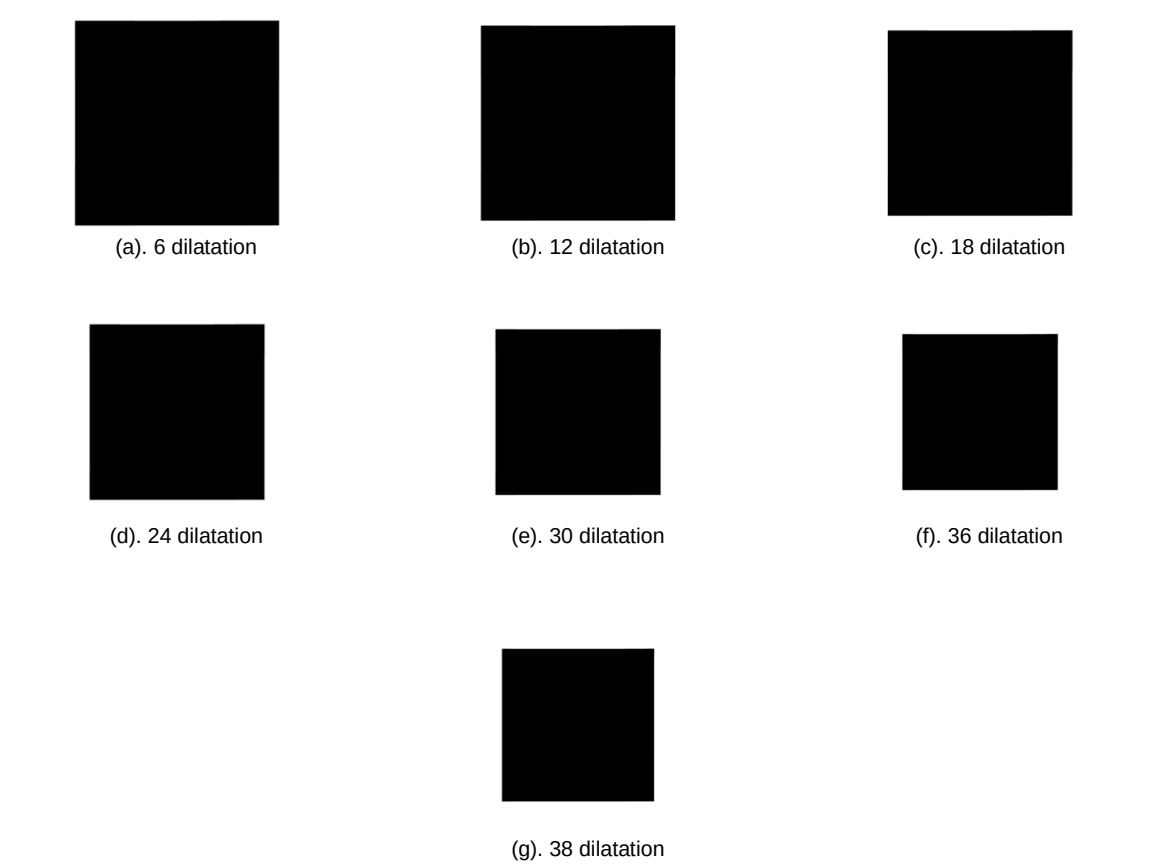


Figure 6: Résultats des dilatations successives du rectangle obtenu par les érosions successives du marqueur visuel.

3. Après obtention d'un rectangle de couleur uniforme et de même taille que celle du marqueur visuel on effectue une opération de détection des contours avec l'algorithme de Canny. Cette opération nous permettra de facilement détecter les lignes grâce à la transformée de Hough qu'on verra par la suite. On obtient le résultat illustré dans la Figure 7 suivante:

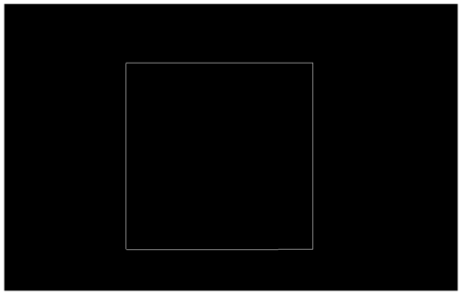


Figure 7: Résultat de l'opération de détection de contours.

Cette opération s'avère inutile vu qu'on doit toujours fixer le nombre d'opérations à effectuer (érosions et dilatation). Pour y remédier, on effectue une seule érosion avec un élément structurant assez grand suivi d'une reconstruction par dilatation jusqu'à arrivée à un critère d'arrêt qui est l'image d'origine du marqueur visuel. Comme ça on n'aura pas à fixer au préalable le nombre d'opérations d'érosion et de dilatation à exécuter. La Figure 8 montre les résultats obtenus:

(a) Image originale du marqueur visuel mais aussi le critère d'arrêt.

(b) Résultat de l'érosion employant un élément structurant assez large.

(c) Résultat après reconstruction par dilatation.

Figure 8: Erosion puis reconstruction par dilatation.

ii. Transformé de Hough

La transformée de Hough est une technique qui peut être utilisée pour isoler les caractéristiques d'une forme particulière dans une image. Comme elle nécessite que les caractéristiques souhaitées soient spécifiées sous une forme paramétrique, la transformée de Hough classique est le plus souvent utilisée pour la détection de courbes régulières telles que des lignes, des cercles, des ellipses … etc.

1. Principe

La transformée de Hough est une technique populaire pour détecter n'importe quelle forme, si on peut représenter cette forme sous forme mathématique. Elle peut détecter la forme même si elle est cassée ou déformée un peu. Nous allons voir comment cela fonctionne pour une ligne.

Une ligne peut être représentée comme $y = mx + c$ ou sous forme paramétrique, comme $\rho = xcos(\theta) + ysin(\theta)$ où ρ est la distance perpendiculaire de l'origine à la ligne, et θ est l'angle formé par cette ligne perpendiculaire et cet axe horizontal comme le montre la Figure 9.

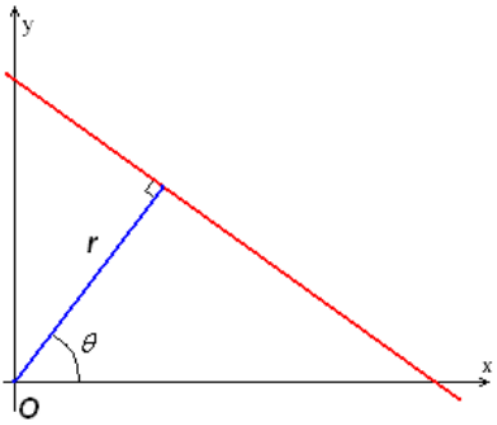


Figure 9: Représentation d'une ligne avec les coordonnées polaires

Voyons maintenant comment Hough Transform fonctionne pour les lignes. Toute ligne peut être représentée par ces deux termes (ρ, θ). Donc, d'abord, il crée un tableau ou un accumulateur 2D (pour contenir les valeurs de deux paramètres) et il est initialement défini sur 0. Soit les lignes dénotent le ρ et les colonnes dénotent le θ. La taille du tableau dépend de la précision dont on a besoin. Supposons qu'on veuille que la précision des angles soit de 1 degré, on a besoin de 180 colonnes. Pour ρ, la distance maximale possible est la longueur diagonale de l'image. Ainsi, en prenant une précision d'un pixel, le nombre de lignes peut être la longueur diagonale de l'image.

Considérez une image avec une ligne horizontale au milieu. Prenons le premier point de la ligne. On connaît ses valeurs (x, y). Maintenant, dans l'équation de la ligne, on met les valeurs $\theta = 0,1,2 \ldots, 180$ et on vérifie le ρ qu'on obtient. Pour chaque paire (ρ, θ), On incrémente la valeur de un dans les cellules (ρ, θ) correspondantes de l'accumulateur.

On prend maintenant le second point sur la droite. On fait la même chose que expliquée précédemment. On incrémente les valeurs dans les cellules correspondant à (ρ, θ) qu'on a obtenues. On peut dire qu'un vote est réalisé sur les valeurs (ρ, θ). On continue ce processus pour chaque point de la ligne. Donc, on recherche le maximum de votes dans l'accumulateur qui nous indiquera qu'il y a une ligne dans cette image dont les coordonnées polaires s'agissent des indices de ce maximum dans l'accumulateur.

2. Application

Après avoir obtenu les contours détectés lors de la dernière étape décrite précédemment, on procède à la détection des lignes par la transformée de Hough sur l'image des contours.

Après avoir effectué la transformée de Hough sur l'image des contours, on peut afficher le contenu de l'accumulateur comme le montre la Figure 10:

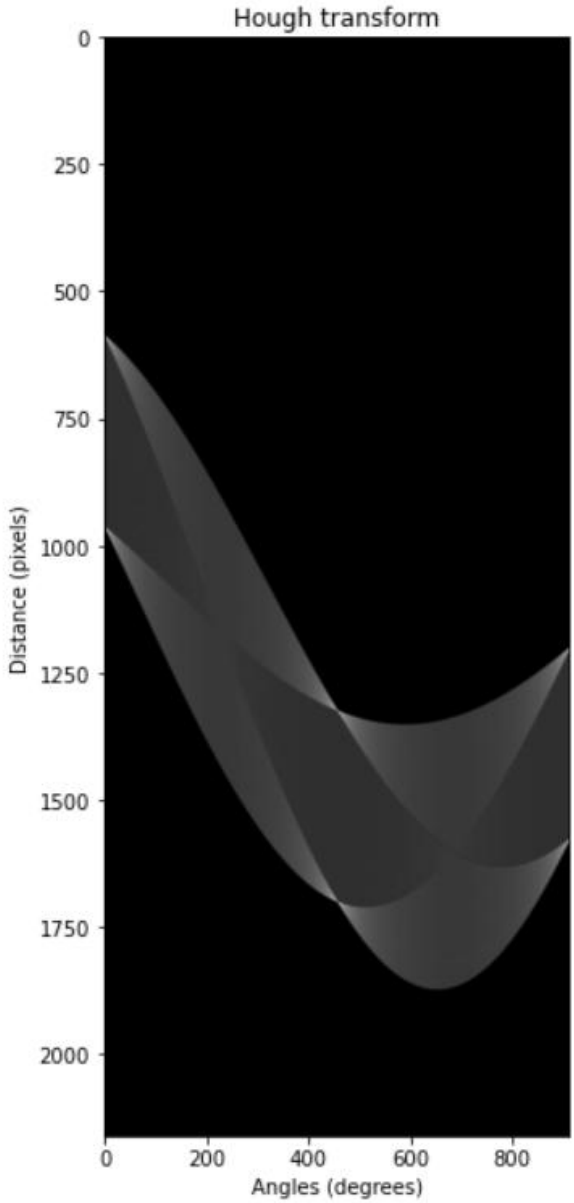
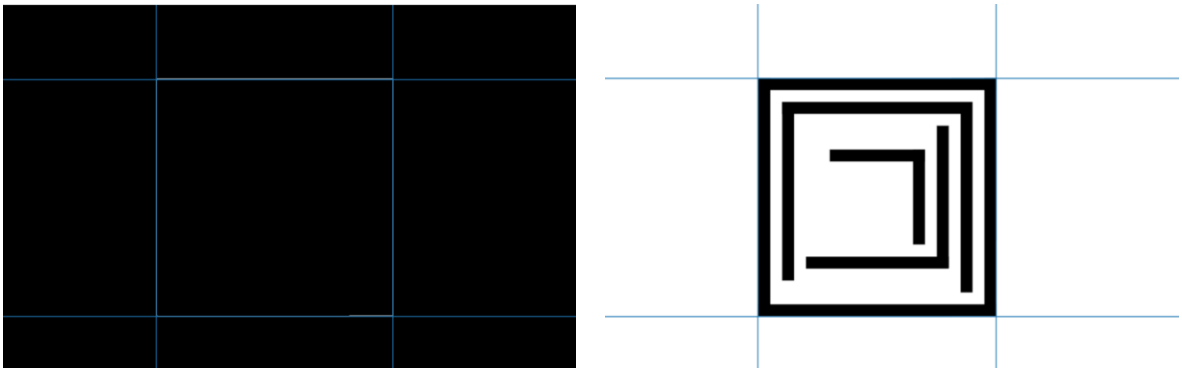


Figure 10: Affichage du contenu de l'accumulateur

On obtient le résultat suivant illustré dans la Figure 11 suivante:



(a) Détection des lignes sur les contours détectés.

(b) Affichage des lignes détectées sur le marqueur visuel.

Figure 11: Détection des lignes par la transformée de Hough

On a pu détecter des lignes qui sont ajustées exactement au marqueur visuel. Après récupération des lignes détectées, on peut maintenant obtenir les coordonnées des quatre coins du rectangle en calculant les coordonnées des points d'intersection de chaque deux lignes et ceci par résolution du système comportant les deux équations des deux lignes respectives.

b. Deuxième approche

i. Clustering

Le clustering consiste à diviser la population ou les points de données en un certain nombre de groupes de sorte que les points de données des mêmes groupes soient plus similaires aux autres points de données du même groupe que ceux des autres groupes. En termes simples, l'objectif est de séparer les groupes ayant des caractéristiques similaires et de les répartir en grappes.

- **K-means**

K signifie est un algorithme de clustering itératif qui vise à trouver des maxima locaux à chaque itération. Cet algorithme fonctionne en plusieurs étapes:

- Pour commencer, nous sélectionnons d'abord un certain nombre de classes/groupes à utiliser et initialisons au hasard leurs points centraux respectifs.

- Chaque point de données est classé en calculant la distance entre ce point et chaque centre de groupe, puis en classant le point comme étant dans le groupe dont le centre est le plus proche de lui.

- A partir de ces points classés, nous recalculons le centre du groupe en prenant la moyenne de tous les vecteurs du groupe.

- Répétez ces étapes pour un nombre défini d'itérations ou jusqu'à ce que les centres de groupe ne changent pas beaucoup entre les itérations.

ii. Application

Puisqu'on cherche à détecter un rectangle dans l'image du marqueur visuel, on peut utiliser l'algorithme K-means (K-moyenne) pour extraire à partir des données fournies de l'image du marqueur visuel quartes clusters ou groupes qui nous permettront par la suite de trouver les quatre coins du rectangle recherché. Pour mettre au point cette méthode, on a utilisé un autre exemple de rectangle pour illustrer cette approche. L'exemple utilisé est montré dans la Figure 12 ci-dessous:

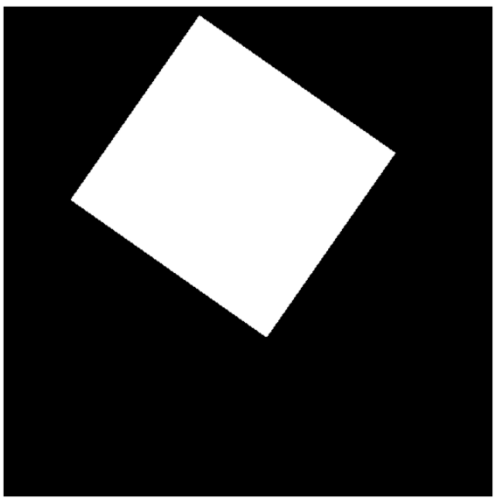


Figure 12: Exemple de rectangle utilisé pour la détection par clustering

Pour cette exemple, on procède à la détection des contours du rectangle, qu'on va par la suite ségréger, grâce à l'algorithme de clustering (K-moyenne), en quatre clusters qui correspondront aux quatre coins du rectangle. Les résultats de cette opération sont illustrés dans la Figure 13 suivante:

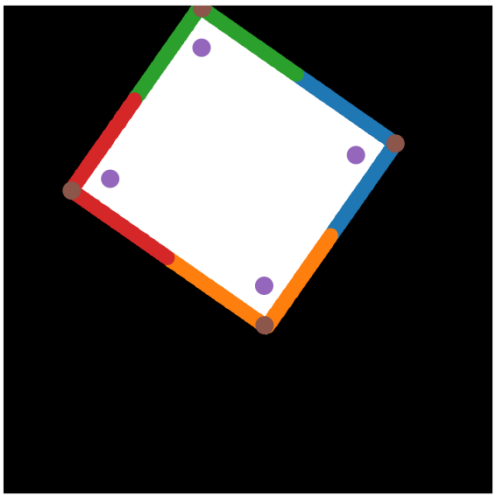


Figure 13: Application de l'algorithme de clustering K-moyenne sur l'exemple donné.

On obtient effectivement quatre clusters chacun représenté par une couleur différente (rouge, bleu, vert, orange) avec un centre de cluster (en violet). À partir de chaque cluster détecté, on peut ensuite trouver les coins qu'on recherche grâce au suivi des changements du gradient des points appartenant à chaque cluster. On obtient donc les coordonnées des quatre coins du rectangle illustré avec une couleur marron sur la Figure 13 précédente.

c. Troisième approche

i. Cartes de distances

La carte de distances est un opérateur appliqué aux images binaires. Le résultat de la transformation est une image en niveaux de gris qui ressemble à l'image d'entrée, sauf que les intensités en niveaux de gris des points à l'intérieur des régions de premier plan sont modifiées pour afficher la distance à la limite la plus proche de chaque point.

ii. Application

On peut appliquer ce concept de carte de distances à notre premier exemple du marqueur visuel OILU. On obtient la visualisation illustrée dans la Figure 14.

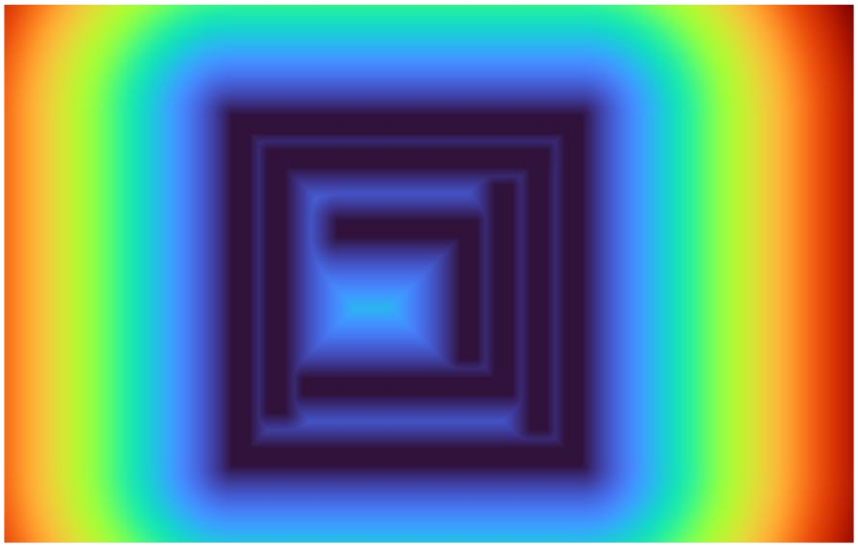


Figure 14: Carte des distances appliquée sur l'image du marqueur visuel

On sélectionne les points dont la distance est égale à 1. On obtient la Figure 15 suivante:

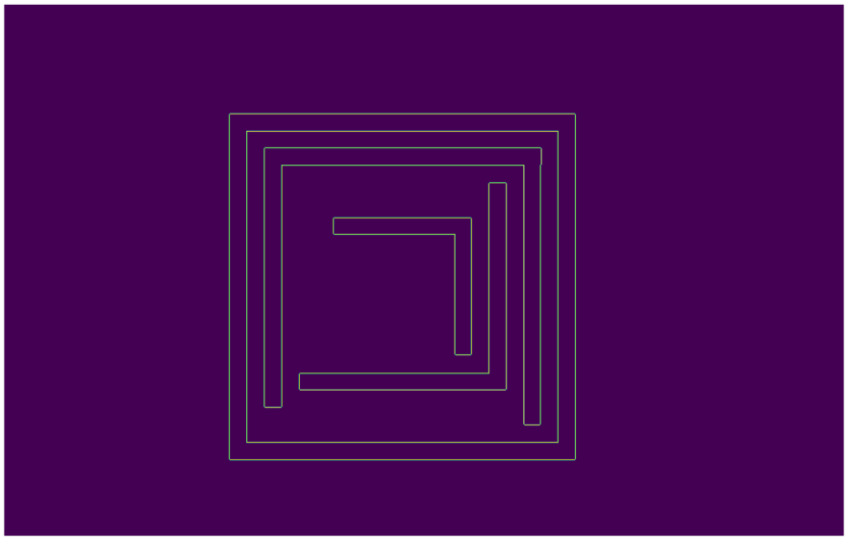
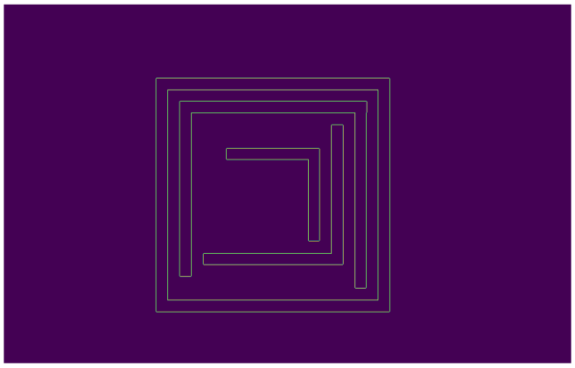
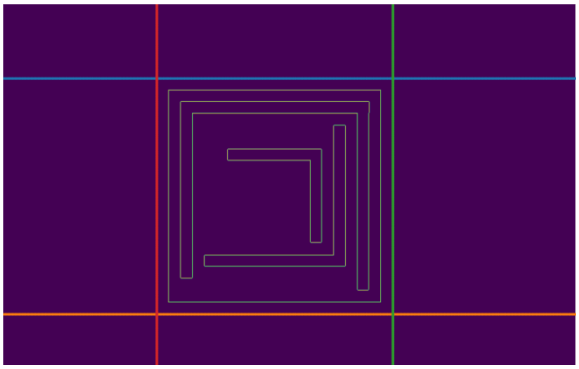


Figure 15: Visualisation illustrant les points de distance égale à 1

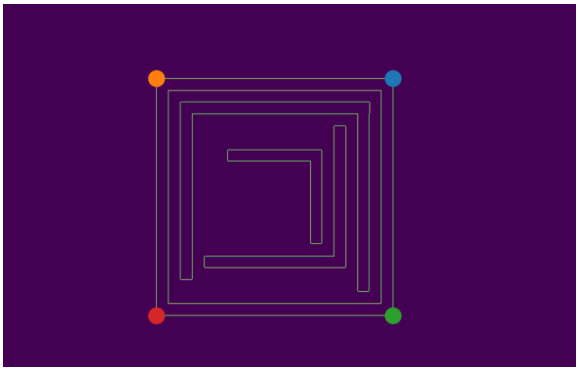
On peut ensuite sélectionner les lignes et colonnes des contours extérieurs en prenant les indices des deux plus grandes sommes des lignes et des deux plus grandes sommes des colonnes. On obtient 4 indices: deux indices des lignes et deux indices des colonnes. Donc on a obtenu les coordonnées des quatre coins du rectangle. Les résultats de ces étapes sont illustrés dans la Figure 16.



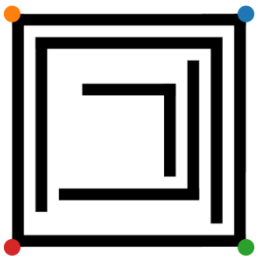
(a) Visualisation des points de distance égale à 1



(b) Lignes et colonnes dont leur somme est la plus larges comparée aux autres lignes et colonnes



(c) Coordonnées des quatre coins du rectangle illustrés sur la visualisation des points de distance égale à 1.



(d) Coordonnées des quatre coins du rectangle illustrés sur l'image du marqueur visuel.

Figure 16: Extraction des quatre coins du marqueur visuel à partir de la carte des distances.

Cependant cette technique d'extraction des coins par les valeurs maximales des sommes des lignes et des colonnes présume que le rectangle doit être bien droit, chose qui n'est pas toujours satisfaite, du coup il est peu probable que cette astuce soit utilisée.

d. Autres approches à envisager

Pour cette étape de détection du marqueur visuel, on a pensé à d'autres approches possibles:

- Calculs des surfaces:

Cette approche consiste à calculer l'ensemble des surfaces présentes sur l'image et de sélectionner celle qui est la plus large qui correspond en fait à la surface du marqueur visuel (en supposant que le marqueur visuel soit de taille assez large et occupant la majorité de l'image).

- Utilisation des réseaux de neurones:

Cette méthode fait intervenir les réseaux de neurones afin de créer et entraîner un modèle de détection des rectangles ou plus précisément des marqueurs visuels. On peut créer un tel modèle de rien ou bien utiliser des modèles déjà existants et entraînés pour d'autre tâches de détection autre que celles des rectangles. Ensuite faire entraîner ce modèle à la tâche de détection des rectangles, mais cette opération d'entraînement requiert un jeu de données pour faire l'entraînement dessus. Ce jeu de données doit être constitué de plusieurs images de rectangles annotées par les coordonnées des différents coins du ou des rectangles. On pense que cette approche serait plus prometteuse et donnera des résultats plus satisfaisant que les autres approches du fait que le modèle apprendra à détecter plus précisément les rectangles sous différentes formes et différentes taille et sera plus robuste aux cas les plus difficiles en supposant que le modèle apprenne à bien généraliser et que le jeu de données soit suffisamment grand et diversifié pour permettre un entraînement plus efficace.

5. Déploiement de l'application sur Android

Pour la mise en place d'une application Android, on a utilisé une librairie open-source sur python appelée Kivy. Cette librairie nous permet de développer des applications Android en langage python.

L'installation de Kivy peut se faire sous n'importe quel système d'exploitation. Sous Windows, cela est fait en quelques étapes:

- Mise à jour de pip, de wheel et de setuptools: en exécutant la commande suivante sur la ligne de commande:

```
python -m pip install --upgrade pip wheel setuptools
```

- Installez les dépendances pré requises:

```
python -m pip installer pygments docutils pypiwin32 kivy.deps.sdl2 kivy.deps.glew
python -m pip installer kivy.deps.gstreamer
python -m pip installer kivy.deps.angle
```

- Finalement, installer Kivy:

```
python -m pip install kivy
```

Pour tester cette librairie, un code affichant le flux de la caméra de l'appareil Android a été développé. L'application doit être composée d'au moins un fichier nommé main.py qui servira par la suite au paquetage et au déploiement de l'application sur l'appareil Android.

Pour déployer ce code sur l'appareil Android, on utilisera la librairie Buildozer qui vise à empaqueter facilement l'application mobile. Buildozer automatise l'ensemble du processus de construction, télécharge les pré requis comme python-for-Android, Android SDK, NDK, etc. Buildozer gère un fichier nommé buildozer.spec dans le répertoire d'application, décrivant les exigences et les paramètres de l'application tels que le titre, l'icône, les modules inclus, etc. Il utilisera le fichier de spécification pour créer un package pour Android, iOS, etc.

Le paquetage et le déploiement de l'application doit se faire sous un environnement Linux. Sous Windows on a plusieurs choix pour avoir un environnement Linux, soit en utilisant une machine virtuelle, soit en utilisant Windows subsystem for Linux qui est nouvelle une fonctionnalité qui permet d'exécuter un environnement Linux directement sur Windows 10, sans aucune modification.

Dès qu'on a un environnement Linux, on procède à l'installation de Buildozer en exécutant dans un terminal les commandes suivantes:

```
sudo apt install git
```

```
git clone https://github.com/kivy/buildozer.git
```

```
sudo apt-get install python3
```

```
sudo apt-get install python3-setuptools
```

```
cd buildozer
```

```
sudo python3 setup.py install
```

Après avoir installé Buildozer on importe le code de l'application à déployer et, sur un terminal également, on se met sur le répertoire contenant le fichier main.py.

On commence par initialiser Buildozer sur ce répertoire. Cela est fait en exécutant la commande suivante:

```
buildozer init
```

Après exécution de la commande précédente, un fichier buildozer.spec est généré. Ce fichier servira à préciser les spécifications de l'application, comme le nom de l'application, les différents fichiers utilisés par l'application, les librairies requises par l'application, les permissions … etc. On aura à modifier ce fichier pour inclure les différentes librairies utilisées dans notre code mais aussi pour donner la permission à l'application d'utiliser la caméra.

Après modification du fichier buildozer.spec, on peut désormais procéder au paquetage et au déploiement de l'application. D'abord on installe quelques librairies et programmes requis pour cette opération. On exécute la commande suivante:

```
sudo apt update
```

```
sudo apt install git zip openjdk-8-jdk python3-pip autoconf libtool pkg-config zlib1g-dev libncurses5-dev libncursesw5-dev libtinfo5 cmake libffi-dev libssl-dev adb
```

```
pip3 install --user --upgrade cython virtualenv
```

```
sudo apt-get install cython
```

On peut dorénavant construire, paqueter et déployer l'application sur l'appareil Android. L'appareil Android doit être en mode développer, être branché et détecté par le programme adb (Android Debug Bridge) installé précédemment. Android doit autoriser l'accès à l'appareil de l'environnement Linux. Toujours dans le même répertoire (là où les fichiers main.py et buildozer.spec sont localisés), On exécute la commande suivante :

```
buildozer Android debug deploy run
```

Cette commande peut prendre du temps pour s'exécuter. À la fin de celle-ci, l'application devrait être construite en déployée sur l'appareil.

À ce point là, il reste encore à incorporer tester les différentes approches discutées précédemment dans l'application mobile mais cela requiert de débugger certaines difficultés.

6. Conclusion

Dans ce projet, on a pu aborder certaines techniques du traitement d'image afin de mettre au point certaines solutions pour la résolution de la tâche de détection du marqueur OILU, mais cela été expérimental et sous certaines conditions optimales comme clarté et la stabilité de l'image du marqueurs, chose qui n'est probablement pas présente dans des cas qui peuvent être plus difficiles à résoudre et requiert davantage d'optimisation afin d'avoir une solution efficace et robuste qui pourrait faire face à n'importe quel image et n'importe quel scénario.

7. Références:

[1]. Messaoud Mostefai, Salah Khodja, Youssef Chahir: *A New Efficient Numbering System : Application to Numbers Generation and Visual Markers Design*. https://arxiv.org/pdf/2103.11727.