

TESTOUD Romain  
AVRIL-TERRONES Thibaud  
Groupe APP5 Info

# Traitement d'images – TP2

## Détection de cercles dans une image

Binôme : Romain TESTOUD et Thibaud AVRIL-TERRONES  
APP5 Informatique – Polytech Paris Sud

## Sommaire :

- 1 – Objectif du projet
- 2 – Technologies utilisées
- 3 – Solution
- 4 – Problèmes rencontrés
- 5 – Compilation et lancement du projet
- 6 – Résultats

# 1 – Objectif du projet

L'objectif de ce projet est d'implémenter grâce à OpenCv et sans utiliser les fonctions déjà existantes, un détecteur de cercles par une méthode cumulative de type Hough.

Le projet est divisé en trois parties :

- **Exercice 1** : Répondre à des questions portant sur les paramètres du détecteur de cercles afin de mieux comprendre comment celui-ci fonctionne
- **Exercice 2** : Implémentation du détecteur de cercles en utilisant OpenCv sur des images tests, puis par la suite sur des images diverses afin de tester le bon fonctionnement de notre programme.
- **Exercice 3** : Analyse des résultats et des performances de notre programme grâce au temps de calcul. Les tests seront effectués sur des images de taille différentes.

L'objectif de ce projet est aussi une introduction à ce qui se fait dans les voitures autonomes, comme par exemple la possibilité de détecter des panneaux de signalisation.

# 2 – Technologies utilisées

Comme pour la première partie, nous avons décidé d'utiliser git afin de sauvegarder notre travail et pour simplifier la remise du projet.

**Adresse du dépôt GIT : <https://github.com/RomsKo9/II-PartTwo.git>**

Nous avons conservé la structure de la première partie, c'est-à-dire l'utilisation de CMake afin de build correctement le projet.

Pour le reste, les technologies utilisées sont les mêmes que pour la première partie, en l'occurrence :

- OpenCV
  - Version 4.1.1
- Xming
  - Génère des fenêtres X des systèmes Linux pour Windows
- C++
- Bash Linux pour Windows

## 3 – Solutions

Pour commencer, nous avons opté pour une solution octroyant une certaine liberté à l'utilisateur. En effet, lors du lancement du programme, l'utilisateur doit renseigner trois paramètres :

- L'image sur laquelle l'utilisateur veut réaliser sa détection
- Le nombre de cercle qu'il souhaite détecter sur l'image
- Le rayon minimal des cercles à détecter

Le nombre de cercles à détecter et le rayon minimal de ceux-ci nous permettent d'éviter les erreurs de détection (par exemple détecter des trop petits cercles qui ne sont pas représentatifs) et de sélectionner les N cercles les plus « importants » afin d'obtenir un résultat satisfaisant.

Au début, nous avons opté pour un seuil de tolérance au niveau de la détection des cercles. Nous comparions les valeurs de l'accumulateurMax (qui s'incrémente quand la valeur de l'accumulateur est un maximum Local) avec un seuil de tolérance comme par exemple : Si la valeur de l'accumulateurMax était supérieure au seuil de tolérance, alors le cercle pouvait être détecté.

Plus le seuil de tolérance était haut, plus le nombre de cercles détectés était grand.

Le problème était qu'il fallait trouver au dixième près la bonne tolérance pour avoir une détection convenable, et ce seuil changeait en fonction de l'image utilisée.

Cette solution ne nous a pas convaincu, car nous avons souvent des résultats incorrects.

La solution finale que nous avons retenue est de créer une structure représentant les cercles à détecter (struct circleToDetect), composée d'une coordonnée (row, col), d'un rayon et surtout d'un score, nous permettant d'obtenir les cercles les plus représentatifs.

Ensuite, nous avons créé une liste de circleToDetect afin de choisir en fonction des paramètres renseignés par l'utilisateur (nbCerclesToDetect et RadiusMini), les nbCerclesToDetect avec le score les plus élevés.

Enfin, on parcourt cette liste composée des cercles les plus représentatifs, et on les affiche sur l'image.

Nous trouvons cette solution satisfaisante, car elle fonctionne bien sur diverses images, même trouvées sur internet, et à un temps de calcul raisonnable (même si nous n'avons pas de référence en terme de temps de calculs pour une détection de cercle).

## 4 – Problèmes rencontrés

Au niveau des problèmes rencontrés, nous nous sommes surtout pris la tête sur des erreurs d'itérations.

En effet, le nombre de boucle for avec des indices variables et très importants et parfois, nous faisons des erreurs sur les indices, ce qui nous a fait perdre pas mal de temps.

Nous avons aussi pas mal buté sur le calcul des maximums locaux, car nous ne parcourions pas le bon nombre de voisins.

Enfin, avant l'implémentation de la solution finale avec le score, nous avons des résultats « random ». En effet, lorsque nous lançons le programme deux fois à la suite, les résultats obtenus n'étaient pas les mêmes, sans trop savoir pourquoi.

Malheureusement, pour des raisons de départ en vacances, nous n'avons pas eu le temps d'implémenter la partie 3.2.

## 5 – Compilation et lancement du projet

Tout d'abord, comme pour la partie 1, il faut s'assurer que Xming soit lancé.



Pour compiler :




- cmake CMakeLists.txt

```
romsko@DESKTOP-1LGEV2:/mnt/c/Users/testo/Desktop/Polytech/5A/TI/TI_PartTwo/TI-PartTwo$ cmake CMakeLists.txt
-- The C compiler identification is GNU 7.4.0
-- The CXX compiler identification is GNU 7.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenCV: /usr/local (found version "4.1.1")
-- Found OpenMP_C: -fopenmp (found version "4.5")
-- Found OpenMP_CXX: -fopenmp (found version "4.5")
-- Found OpenMP: TRUE (found version "4.5")
-- Configuring done
-- Generating done
-- Build files have been written to: /mnt/c/Users/testo/Desktop/Polytech/5A/TI/TI_PartTwo/TI-PartTwo
```

- make

Pour lancer :

```
/TI-PartTwo$ ./bin/detect_cercle images/jo.png 5 30
```

-  : Image pour la détection de cercle
-  : Nombre de cercles à détecter
-  : Rayon minimal (en pixels) des cercles que l'on souhaite détecter

## 6 – Résultats

```
#####
#                                     #
# Romain TESTOUD & Thibaud AVRIL-TERRONES #
#                                     #
#####

Détection de cercles sur images/jo.png en cours ...

##### RESULT #####

Fichier chargé : images/jo.png
Nombre de cercle(s) à détecter : 5
Rayon de détection minimal : 10 px
-----
Temps d'exécution : 19.9375 s
-----
```

