

Rapport de Perception

Yago Milagres Passos

2 novembre 2025

1 Introduction et Objectif

L'intention du travail est de composer un algorithme qui permet de tracer l'évolution de la position d'une balle dans une vidéo. Pour cela, il suffit de réaliser des actions et traitements de repères (images) pour trouver le mouvement.

1. Clicker et selectioner le pixel qui donne le couleur dela balle ;
2. Changer le couleur pour le dimension HSV ;
3. Appliquer un mask pour marquer seulement les valeurs HSV seletctioné, essayant de effacer les autres elements de l'image ;
4. Appliquer un nettoyage de **morphologie**, s'agit des erosions et dilatations morphologiques en sequence, l'ordre de ces actions peut resulter dans un ouverture ou un fermeture.
5. Appliquer un gradient, pour obtenir juste la circonference de la balle.
6. Submettre l'image resultat sur le fonction HoughCircles, pour trouver les formats circulaires de l'image.

2 Développement et Méthodologie

Le développement de l'algorithme a principalement reposé sur la **documentation officielle en ligne d'OpenCV**, reconnue pour son guide très précis et efficace. J'ai également utilisé l'outil d'IA **Google Gemini** pour des aides ponctuelles, notamment pour des problèmes de syntaxe ou de logique dans la manipulation des tableaux NumPy.

2.1 Sélection de Couleur et Masquage (HSV)

La première phase a consisté à isoler la balle en sélectionnant sa couleur par un clic de souris. Les premières tentatives utilisant l'espace de couleur **RGB** se sont avérées imprécises et devaient être rapidement abandonnées. J'ai constaté que ce modèle **couple l'information de couleur (teinte) avec l'intensité lumineuse** (voir Figure 1), le rendant trop sensible aux variations de luminosité (ombres et reflets).

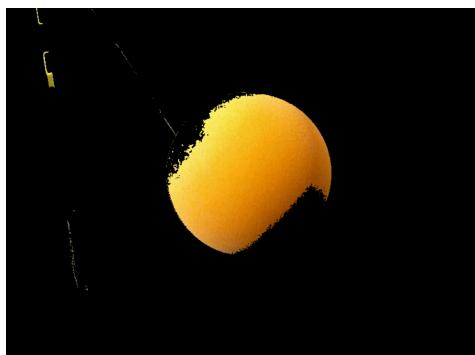


FIGURE 1 – Masque en RGB.

J'ai donc basculé vers l'espace **HSV** (Teinte, Saturation, Valeur). Ce modèle est préféré en segmentation d'objets, car il **sépare la Teinte (H)**, qui définit la couleur, de la **Saturation (S)** et de la **Valeur (V)**, qui encodent la pureté et la luminosité. Le masquage a été appliqué en calculant l'intervalle de la teinte (H) de manière circulaire, pour gérer le passage du 180 (pour les couleurs rouges), tout en appliquant des tolérances sur la saturation et la valeur.

Initialement, avant l'ajustement des tolérances, le masque couvrait trop de zones (voir Figure 2.1). Après un réglage fin des tolérances (voir Figure 2.1), j'ai obtenu un masque initial satisfaisant.

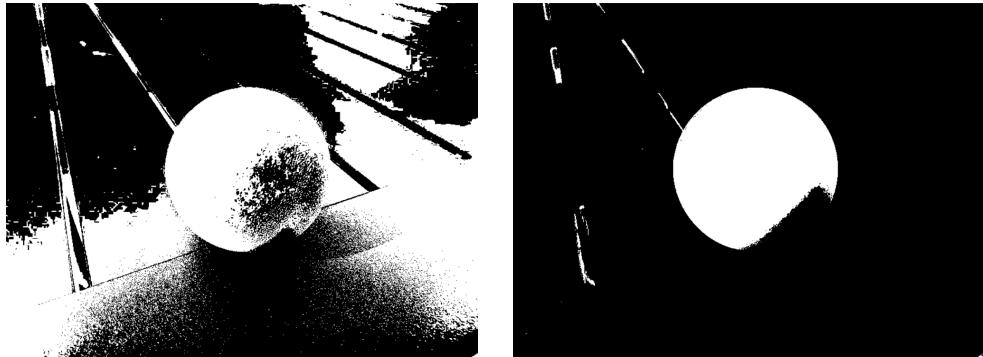


FIGURE 2 – Comparaison du masque avant (2.1) et après (2.1) les opérations morphologiques.

2.2 Séquence de Travail Erronée et Détection Initiale

À ce stade, ma décision initiale fut de sauter l'étape de pré-traitement morphologique pour appliquer immédiatement la fonction `HoughCircles`. J'ai passé un temps considérable à ajuster empiriquement les paramètres de la fonction, n'ayant pas un masque suffisamment traité. Les paramètres trouvés étaient les suivants :

```
circles = cv.HoughCircles( blur , cv.HOUGH_GRADIENT, 1, 20,
                           param1=100, param2=20,
                           minRadius=1, maxRadius=0)
```

Avec ce peu de traitement, les résultats étaient parfois encourageants :

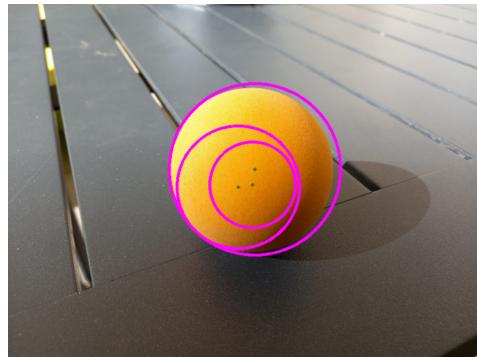


FIGURE 3 – Résultats favorables obtenus après un ajustement initial des paramètres.

Toutefois, ce processus était très instable. Le simple fait de ne pas cliquer parfaitement au centre de la balle, ou un léger changement de luminosité, entraînait une détection incorrecte et l'apparition de nombreux cercles parasites :

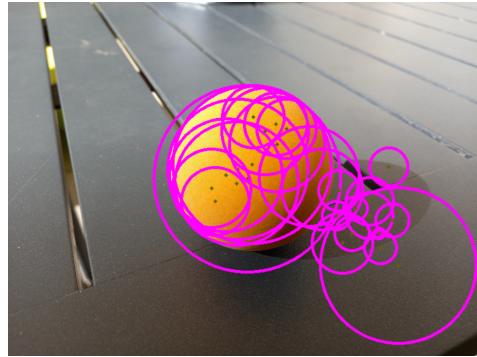


FIGURE 4 – Multiples cercles parasites retournés par la fonction.

J'ai mis en place une solution palliative en créant un algorithme pour systématiquement sélectionner le cercle possédant le rayon le plus grand, partant du principe que la balle serait l'objet circulaire le plus volumineux. Cela a donné des résultats précis dans certaines conditions :

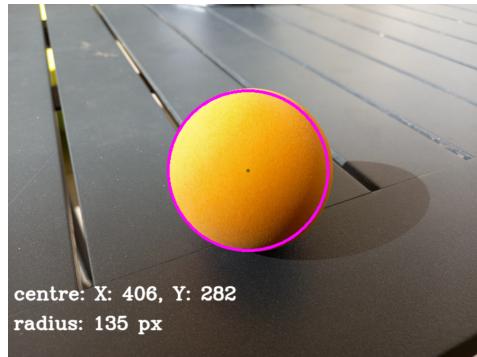


FIGURE 5 – Sélection du cercle au rayon maximum (Résultats précis).

Cependant, j'ai réalisé que l'algorithme n'était pas idéal et comportait des failles (précision du clic requise, mauvaise gestion de la luminosité du HSV), le rendant inutilisable pour l'objectif final du travail : l'application dans un flux vidéo. Le résultat d'un échec de filtrage menant à une détection erronée est illustré ci-dessous :

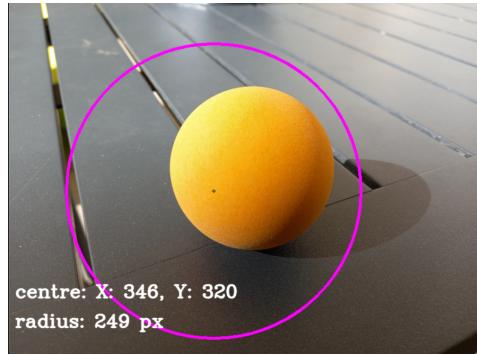


FIGURE 6 – Résultat imprécis.

3 Amélioration du Prétraitement et Suivi

3.1 Restructuration et Morphologie

J'ai pris la décision de restructurer mon code en fonctions distinctes pour le masquage, la morphologie et la détection. Suite à l'enseignement sur les opérations morphologiques, j'ai appliqué le traitement et tout s'est grandement amélioré. La documentation d'OpenCV a fourni un traitement morphologique efficace, consistant en une **ouverture** (érosion puis dilatation) suivie d'une **fermeture** (dilatation puis érosion). J'ai augmenté le nombre d'itérations à 5 pour obtenir des résultats plus satisfaisants.

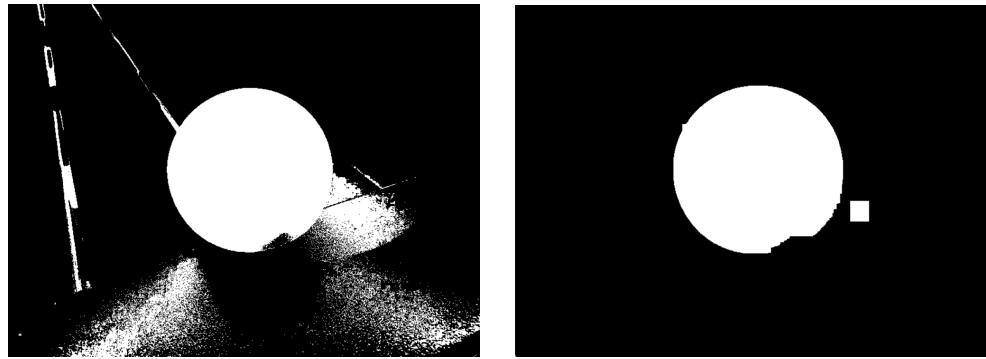


FIGURE 7 – Comparaison du masque avant (3.1) et après (3.1) les opérations morphologiques.

3.2 Gradient et Détection Fiable

Après la morphologie, l'application du gradient est devenue simple. J'ai utilisé la fonction `cv.Laplacian` pour obtenir le contour précis de la balle :

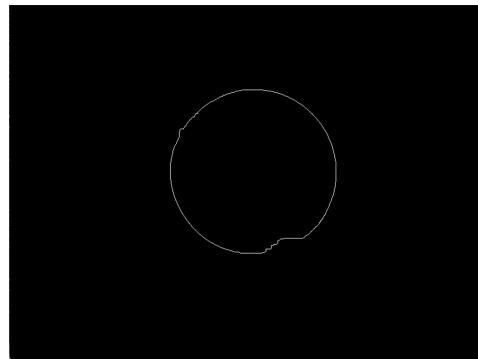


FIGURE 8 – Contour obtenu avec le gradient Laplace.

Avec une image aussi polie, je suis revenu à la fonction `HoughCircles`. Il est devenu très simple de trouver le cercle, et je n'ai plus eu besoin de la solution palliative. Le premier cercle retourné par la fonction fonctionnait à 100% des fois sur l'image statique :

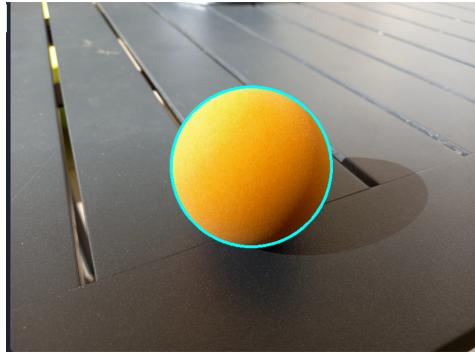


FIGURE 9 – Résultat de la détection fiable.

4 Application Vidéo et Optimisation

4.1 Traçage Initial et Problème de Performance

J'ai ensuite appliqué le processus en vidéo. L'approche consistait à mettre la première image en pause pour la sélection HSV, puis à traiter chaque trame de manière séquentielle pour former la trajectoire (motion track) :

```
first_mask = apply_mask(first_frame, first_color_hsv)
first_morph = apply_morphologie(first_mask)
first_gradient = cv.Laplacian(first_morph, cv.CV_8UC1)
first_circle = findGoodCircle(first_gradient)
```

Cependant, à cause de la quantité de traitement nécessaire, la lecture vidéo est devenue lente (faible FPS).

4.2 Solution par Région d'Intérêt (ROI)

Pour résoudre le problème de performance, j'ai suivi la recommandation de créer une **Région d'Intérêt (ROI)**. Une ROI est une fenêtre de traitement plus petite. J'ai donc créé un repère de taille 600 (`roi_size=600`) et tenté d'appliquer le traitement uniquement dans cette zone.

Initialement, j'ai rencontré des problèmes d'offset car mon ROI était fixé aux coordonnées de la trame entière et non au centre de la balle. J'ai donc traité le code pour **centrer la ROI sur le centre du dernier cercle détecté** par `HoughCircles`. Ceci a fonctionné parfaitement, car cela réduisait la zone de recherche tout en garantissant que la balle était toujours à l'intérieur de la fenêtre.

4.3 Ajustements Finaux

Alors que la détection de mouvement était presque parfaite, un problème est survenu lorsque la caméra s'approchait de la balle dans une partie de la vidéo, entraînant une perte de précision de détection.

J'ai effectué deux ajustements finaux :

1. J'ai augmenté la taille de la ROI à 800 (`roi_size=800`).
2. J'ai observé que le masque devenait plus faible lorsque la balle était proche (car l'éclairage changeait légèrement), j'ai donc augmenté les tolérances HSV pour compenser cette variation.

Ces ajustements ont permis d'obtenir un suivi stable et précis :

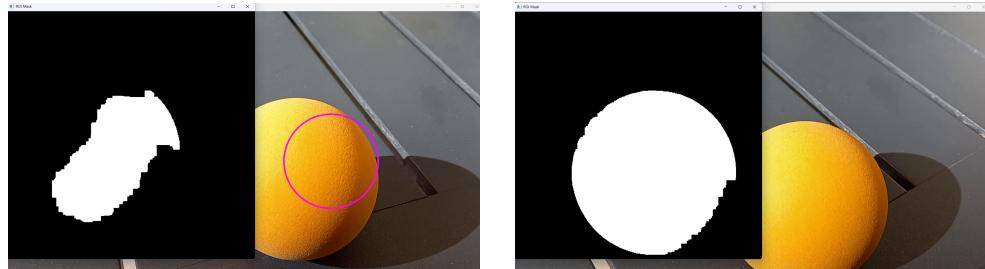


FIGURE 10 – Amélioration de masque du suivi vidéo.

5 Conclusion

L'algorithme final a permis d'atteindre l'objectif fixé. Grâce à la combinaison d'un pré-traitement morphologique efficace et de l'utilisation d'une Région d'Intérêt (ROI) dynamique, la détection des cercles est devenue stable et le traçage de la balle précis.