UNIVERSITÉ DE LIMOGES
Faculté des Sciences et Techniques

Année Universitaire 2016 - 2017

Master INFORMATIQUE ISICG, semestre 2

TRAITEMENT NUMÉRIQUE DES IMAGES

DÉTECTION DE CIBLES DE FLÉCHETTES

Rapport de projet

version du 3 mai 2017

Auteur

ROUIJEL Mehdi

Responsable : M. CRESPIN Benoît

INTRODUCTION

L'objectif de ce projet est de mettre en pratique les notions vues en cours, en particulier la transformée de Hough, en créant une application capable de détecter une cible de fléchettes dans une image. Au-delà de la simple détection de la cible, le programme devrait pouvoir reconnaître ses différentes zones ainsi que les numéros autour ; si des fléchettes sont présentes dans l'images, elles devraient être détectées aussi, pour éventuellement être capable de déterminer le score.

I Utilisation de l'application

Une fois l'application lancée depuis Eclipse, une image par défaut est chargée. Pour utiliser une autre image, il suffit de naviguer vers « File > Open... » pour sélectionner une nouvelle image. Il est aussi possible de sauvegarder l'image qui est vue dans la partie droite de la fenêtre, après avoir exécuté une opération, en utilisant « File > Save... ». La copie (« Edit > Copy ») n'est pas implémentée.

Chaque bouton du panneau de contrôle est indépendant. Par exemple, il est possible de n'appliquer que le flou, mais c'est l'image originale, et non le résultat de celui-ci, qui sera utilisée si la détection des contours est appliquée ensuite.

Différents paramètres peuvent être changés, en ouvrant la barre d'options du panneau de contrôle, pour la détection de contours, la détection de lignes et la détection d'ellipses ; elles sont expliquées dans la suite de ce rapport. Modifier ces paramètres aura un effet important sur le résultat final.

II Détection des contours

La détection des contours se fait par une application du filtre de Sobel qu'il est possible d'améliorer grâce aux différentes options disponibles dans le panneau de contrôle. Sélectionner l'ensemble des options rapproche le traitement d'un filtre de Canny[1], avec dans l'ordre :

1. Réduction du bruit par un filtre de Gauss
2. Seuillage
3. Analyse de région (« blob analysis »)
4. Amincissement

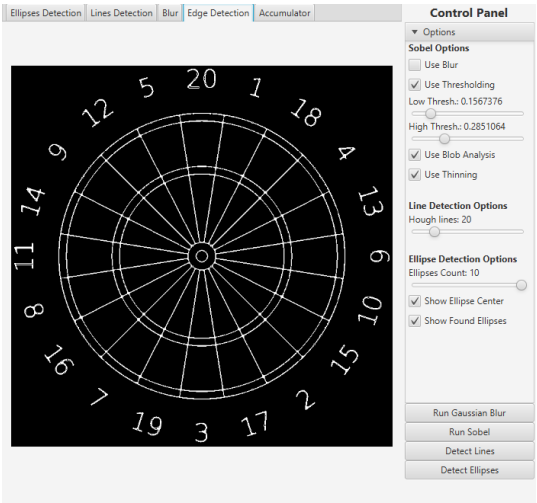

FIGURE 1 – Contours trouvés

Il est à noter que deux seuils peuvent être modifiés. Le seuil haut concerne l'étape de seuillage et détermine la valeur frontière pour séparer les pixels affichés des pixels de fond. Le seuil bas est utilisé par l'analyse locale ; celle-ci ne pourra d'ailleurs fonctionner que si le seuillage est aussi sélectionné.

Le fonctionnement de l'analyse locale est simple : lors de l'étape de seuillage, les pixels au-dessus du seuil haut sont marqués comme « forts » et les pixels entre les deux seuils sont marqués comme « faibles ». Par la suite, un pixel faible ne sera gardé que s'il est connecté à un pixel fort. En fonction de l'image, nous pouvons constater une amélioration du résultat.

III Détection des lignes

Le programme utilise la transformée de Hough pour détecter les lignes.

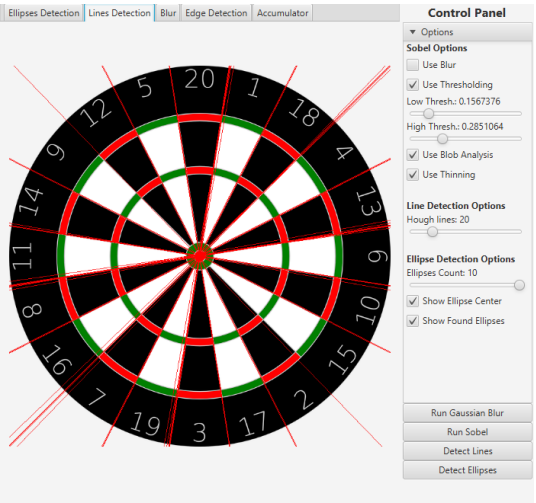

FIGURE 2 – Lignes trouvées

Dans l'image obtenue par la détection de contours, on regarde les i pixels blancs. Chacun de ces pixels appartient aux droites à une distance ρ_i de l'origine, et orientées à un angle θ_i sur l'intervalle $[0..2\pi]$; dans une matrice de taille $\rho_{max} \times \theta_{max}$, nous incrémentons alors la case correspondante. Plus il y a de pixels le long d'une droite, plus la valeur de la case (ρ, θ) associée sera élevée, et donc plus il est probable que nous ayons détecté une ligne.

Il ne nous reste plus qu'à récupérer les n valeurs les plus élevées pour tracer les n droites détectées en repassant du système polaire au système cartésien grâce aux équivalences $x = \rho \cdot \cos\theta$ et $y = \rho \cdot \sin\theta$.

IV Détection des ellipses

Dans le cas de cibles de fléchettes, la majorité des lignes trouvées devraient être les délimitations des sections qui convergent vers le centre. Cependant, la détection de lignes n'est jamais exacte, nous aurons donc plusieurs points d'intersection proche du « vrai » centre.

Pour déterminer le centre qui sera utilisé, nous regardons les lignes deux à deux et mémorisons leur intersection. Une procédure en « force brute » détermine ensuite les deux intersections les plus proches, puis le centre choisi sera la moyenne de ces deux intersections.

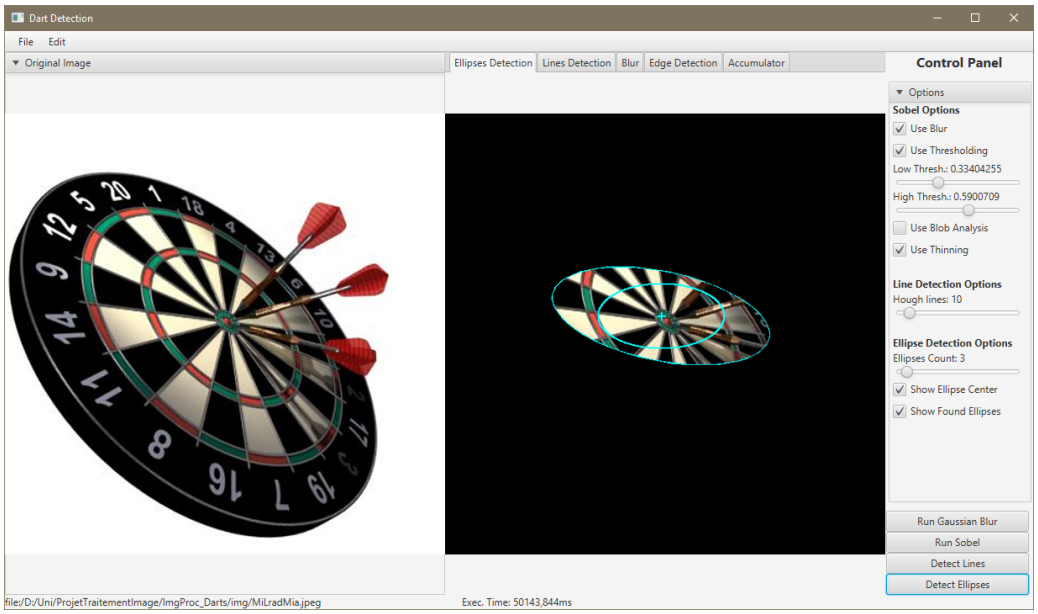

FIGURE 3 – Résultat non convaincant

À partir de ce centre, les combinaisons possibles de rayons et d'angles de rotation sont regardées et comparées avec l'image issue de la détection de contours. C'est, là aussi, un procédé en « force brut » qui est relativement lent pour un résultat très discutable, comme le montre la figure 3.

Le résultat obtenu peut néanmoins être amélioré en jouant avec les paramètres, et est acceptable dans le cas d'une cible vue de face et suffisamment éclairée (c.f. Figure 4).

V Travail non réalisé

Par manque de temps, la détection des nombres, des sections de la cible et des fléchettes prévues pour ce projet ne sont pas implémentées.

La détection des sections nécessiterait une détection des ellipses plus précise qui, combinée avec la détection de lignes, pourrait permettre un étiquetage des zones trouvées.

La détection des nombres et des fléchettes sont plus complexes et demanderaient plus de réflexion.

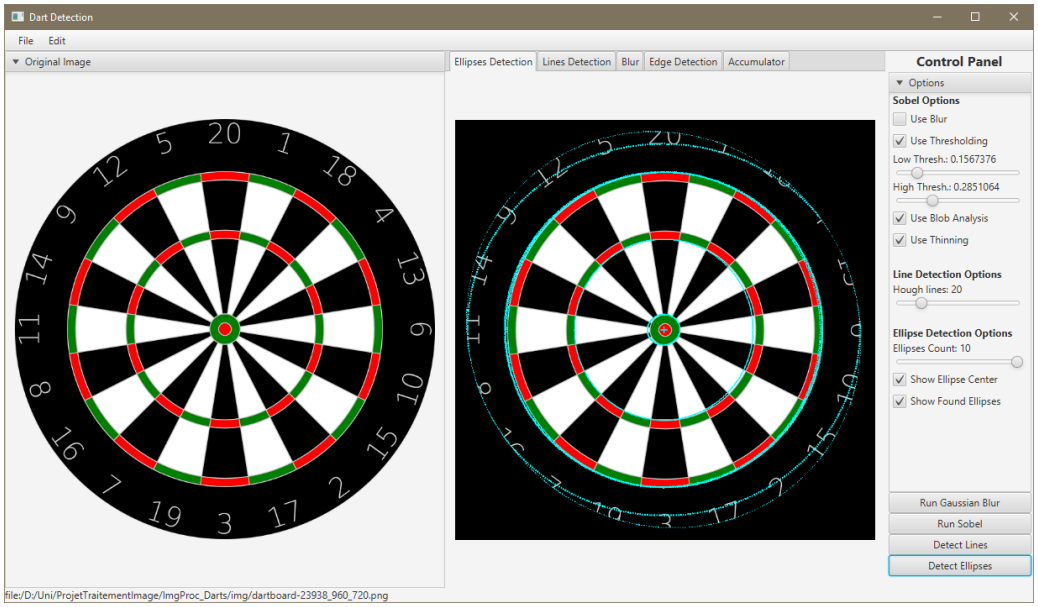

FIGURE 4 – Résultat de la détection d'ellipses avec 1 et 2

CONCLUSION

Les résultats obtenus ne sont pas satisfaisants. Au-delà des fonctionnalités manquantes, il serait possible d'améliorer le code existant à plusieurs niveaux.

Sur le plan de la conception globale de l'application, nous pourrions procéder à quelques ajustement ; elle suit assez librement le patron de conception MVC et son implémentation pourrait être plus stricte.

Ensuite, les procédures en « force brute » évoquées plus haut dans le rapport pourraient être remplacées. Pour la recherche des points les plus proches, notamment, il existe plusieurs algorithmes qui permettraient de réduire le temps d'exécution, surtout pour un nombre de lignes élevé.

Enfin, avec une détection d'ellipses plus précise, il serait possible de déterminer si l'objet détecté est bien une cible, par exemple grâce au nombre d'ellipses concentriques. En l'état, l'application ne serait pas capable de faire la différence entre une cible et une roue de vélo.

Références

[1] Canny edge detector, 2017. Article Wikipedia.

[2] Hough Transform - Isolating ellipses, 2016. Wiki du programme MIPAV.

[3] Y. Xie and Q. Ji. A NEW EFFICIENT ELLIPSE DETECTION METHOD, 2002. Publication.

[4] A. Vasiliauskas. Ellipse detection in image by using Hough transform, 2011. Article de blog.

[5] Image convolution, 2016. Page web.

[6] HoughTransform.java, 2017. Page web de l'université d'Essex.

[7] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. Hough Transform, 2003. Page web.

[8] Solutions for y for a rotated ellipse, 2010. Fil de discussion StackOverflow.