TIPE

Partie 3 : Recherche par couleur

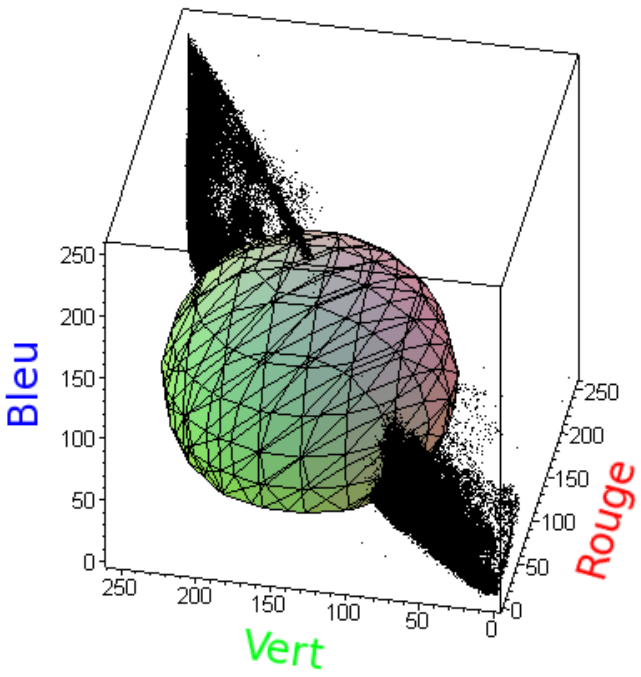
Ayant essayé d'implémenter une algorithme pour détecter une balle par sa forme circulaire, avec des résultats variés et pas toujours satisfaisants, et au vu des problèmes de détection dus à l'imprécision de la détection de contours, j'ai essayé de construire moi-même un algorithme de détection, non plus géométrique, mais se fondant sur la couleur jaune particulière de la balle, et visant à améliorer les performances de l'algorithme précédent dans mon cas très particulier d'une balle de tennis.

1. Filtrage des couleurs

Ma première idée a été de prendre quelques photos de balles sous différentes luminosités, et de regarder quelles étaient les **couleurs principalement présentes** dans les balles pour éliminer les autres de l'image.

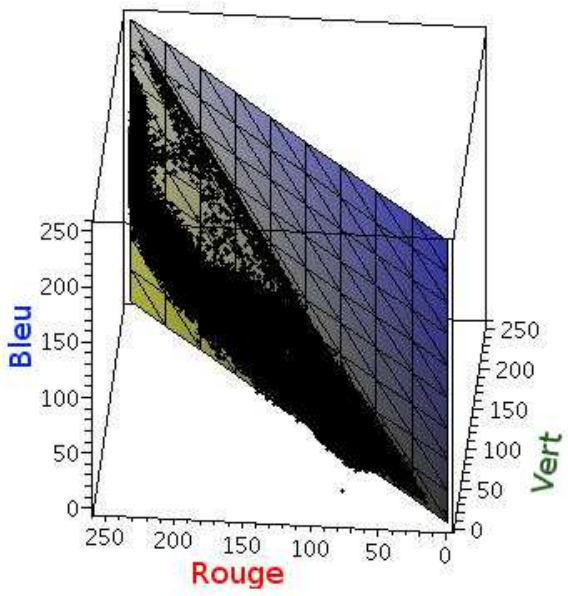
Je noterai une couleur par $c = (r,\ g,\ b)$, avec r,g,b les proportions de rouge, vert et bleu de la couleur (entre 0 et 255), et $d(c_1,\ c_2) = \sqrt{(r_1-r_2)^2+(g_1-g_2)^2+(b_1-b_2)^2}$ la distance usuelle dans l'espace RGB entre deux couleurs. Après analyse avec Maple, j'ai obtenu une couleur moyenne et un écart-type de la distance à la couleur moyenne de :

$\bar{c_0} = (154, 154, 51)$
$\sigma_0 = 98, 27$



Cet écart-type, comme on le voit sur l'image (la sphère est centrée autour de $\bar{c}$ de rayon σ), est cependant trop important pour pouvoir simplement enlever les couleurs qui sont situés trop *loin* de la couleur moyenne : il faut trouver une autre méthode, plus précise.

Or, sur l'image, la zone de l'espace RGB où se trouvent les couleurs de la base est en réalité approximativement un triangle dans un plan. Le sens commun nous dit que rouge + vert = jaune, et, en pratique, cela est prouvé par une régression linéaire avec MAPLE.

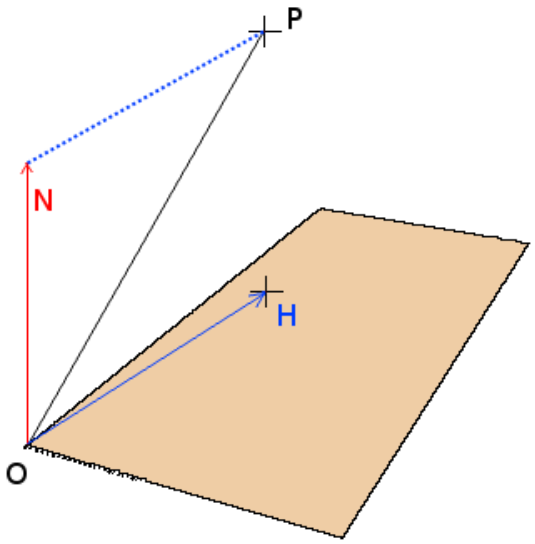


L'**équation du plan** dans lequel se trouvent les couleurs *jaunes*, que je noterai dans la suite $\mathcal{P}$ est donc $r-g=0$. Un vecteur normal unitaire est $\vec{m}=\frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\\0\end{pmatrix}$.

La droite qui sépare la partie *jaune* du plan, celle qui nous intéresse, de la partie *bleue* est la droite d'équation $\begin{cases}r=t\\g=t\\b=t\end{cases}$, de vecteur directeur $\vec{\delta}=\begin{pmatrix}1\\1\\1\end{pmatrix}$. Un vecteur normal à cette droite, dans $\mathcal{P}$, est $\vec{n}=\begin{pmatrix}\alpha\\\beta\\\gamma\end{pmatrix}$, avec $\begin{cases}\vec{n}\in\mathcal{P}\\\vec{n}.\vec{\delta}=0\end{cases}\Leftrightarrow\begin{cases}\alpha-\beta=0\\\alpha+\beta+\gamma=0\end{cases}$. Dans ce cas, $\vec{n}=\begin{pmatrix}1\\1\\-2\end{pmatrix}$ convient.

Il apparaît alors qu'**un point C du plan $\mathcal{P}$ est dans le triangle jaune, si et seulement si** $\vec{n}.\overrightarrow{OC}>0$.

Enfin, il est possible de filtrer les points qui sont trop loin pour être jaunes.



Comme on le voit, $\overrightarrow{OP} = \overrightarrow{ON} + \overrightarrow{OH}$. **La distance de P au plan est** $d = \|\overrightarrow{ON}\| = \vec{m}.\overrightarrow{OP}$ ($\vec{m}$ est unitaire).

A présent, nous pouvons définir un premier filtre sur les points de l'image : soit c une couleur. Soit $d_{\max,y}$ la distance maximale d'une couleur au plan jaune. On peut définir $c' = t(c)$ par :

$$t(c) = \begin{cases} \text{bleu}(0,0,255) & \text{si } \vec{n}.c < 0 \\ \text{bleu}(0,0,255) & \text{si } \vec{n}.c > 0 \text{ et } \vec{m}.c > d_{\max,y} \\ c - (\vec{m}.c)\vec{m} & \text{si } \vec{n}.c > 0 \text{ et } \vec{m}.c < d_{\max,y} \end{cases}$$

En appliquant t à l'image initiale, on obtiendra une image dont les points seront **bleus** (une couleur arbitraire, mais suffisament distincte du jaune) si ils sont trop différents du jaune, et d'une **teinte de jaune** correspondant à leur projection sur le plan jaune sinon. Ce filtrage élimine déjà un grand nombre de points à traiter.

Image initiale

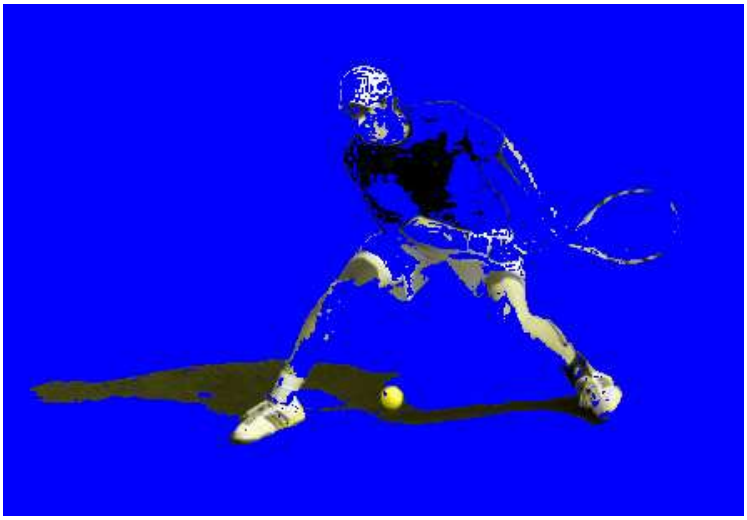


Image projetée ($d_{\max,y} = 35$)

2. Regroupement en composantes connexes

Après avoir supprimé de l'image les points qui n'appartiennent clairement pas à la balle, ce qui m'a paru une des manières les plus efficaces de détecter la balle est le regroupement en composantes connexes. En effet, dans une image aux conditions défavorables, la plupart des points de couleur jaune n'appartiennent pas à la balle, mais à d'autres éléments du décor, qui ne sont pas forcément de la bonne couleur, ni de la bonne taille pour pouvoir être la balle. J'ai donc pensé à trouver **la composante connexe la plus petite à laquelle ces points appartiennent**, où la connexité est définie d'une manière un peu spéciale : en effet, un point est dans la même composante qu'un de ses voisins si la couleur de ce point est proche de la couleur de la composante à laquelle appartient le voisin en question. C'est une manière de garantir que la composante connexe reste d'une couleur à peu près uniforme, et de s'assurer que la composante connexe de la balle ne contiendra qu'elle, tout comme les autres composantes du décor ne contiendront que des éléments du décor.

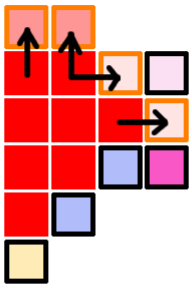
A partir d'une liste de points (parfois appelés "graines" – *seeds* en anglais – dans la littérature), je voulais donc obtenir des composantes connexes auxquelles appartiennent chacun de ces points. L'algorithme que j'ai utilisé, appliqué à un point, fonctionne en **s'étendant par contagion** (d'où le nom de "graines" donné aux points initiaux) jusqu'à ce que plus aucun point ne puisse être ajouté dans la composante connexe :

```
tant que (il reste des points à traiter) faire
  extraire le prochain point à traiter
  pour chaque voisin du point à traiter faire
    si il peut être ajouté dans la composante connexe
      alors l'ajouter dans la liste des points à traiter, et dans la composante connexe
  fin faire
```

```
fin faire
renvoyer la composante connexe
```

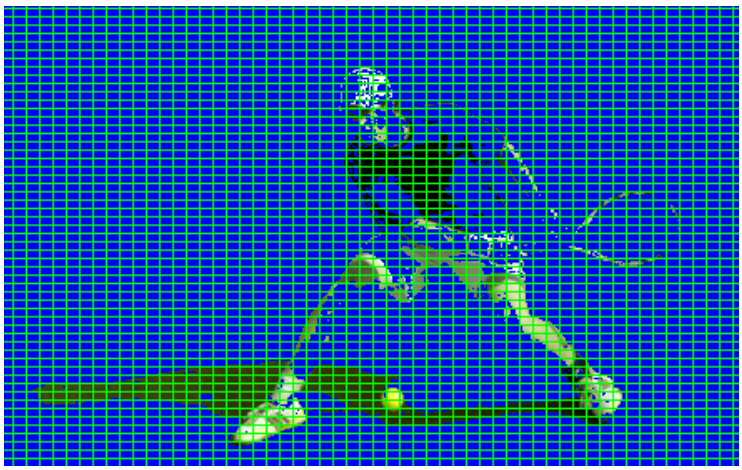
Cet algorithme est très général ; pour rajouter ma condition d'**unformisation de la couleur**, j'ai dû modifier la condition d'ajout dans la composante. Tout au long de l'expansion de la composante connexe, la couleur moyenne $\bar{\gamma}$ de celle-ci est recalculée en permanence, et je n'ajoute un point de couleur γ dans la composante actuelle que si $d(\gamma, \bar{\gamma}) \leqslant d_{\max,c}$, avec $d_{\max,c}$ donnée.

De plus, un point n'est ajouté que s'il n'appartient pas déjà à une autre composante connexe, c'est-à-dire si **les composantes connexes restent disjointes deux à deux**, ce qui permet de réduire de manière significative le temps d'exécution.

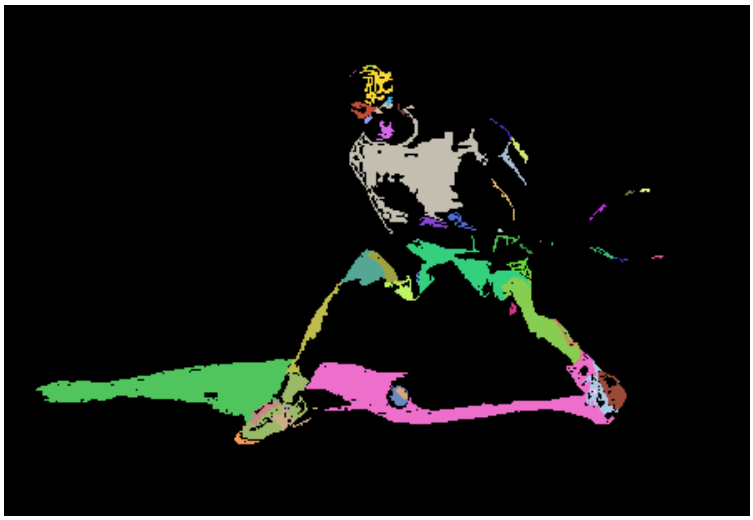


Extension de la composante connexe vers les voisins de couleurs proches

J'obtiens donc une **segmentation de l'image** en plusieurs composantes, dont l'une d'elle sera peut-être la balle. Pour initialiser les graines, j'ai d'abord pensé à des points les plus proches de la couleur moyenne de la balle, calculée précédemment (cf. 1), mais il est vite apparu que de très nombreux points proches appartenaient à la même composante, et que l'image n'était pas entièrement parcourue. J'ai donc modifié l'algorithme pour que les points de départ forment une grille recouvrant toute l'image, ce qui permet de s'assurer que des composantes redondantes n'apparaissent pas. On notera g_h et g_w les paramètres verticaux et horizontaux de la maille de la grille.



Grille de points initiaux ($g_h = 55$, $g_w = 55$)

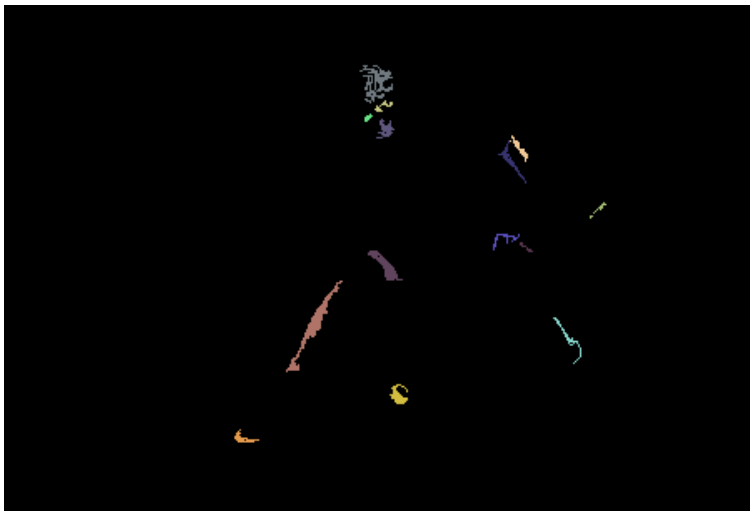


Segmentation de l'image projetée ($d_{\max,c} = 42$)

3. Elimination des composantes connexes

A présent, il s'agit d'éliminer les composantes connexes une à une.

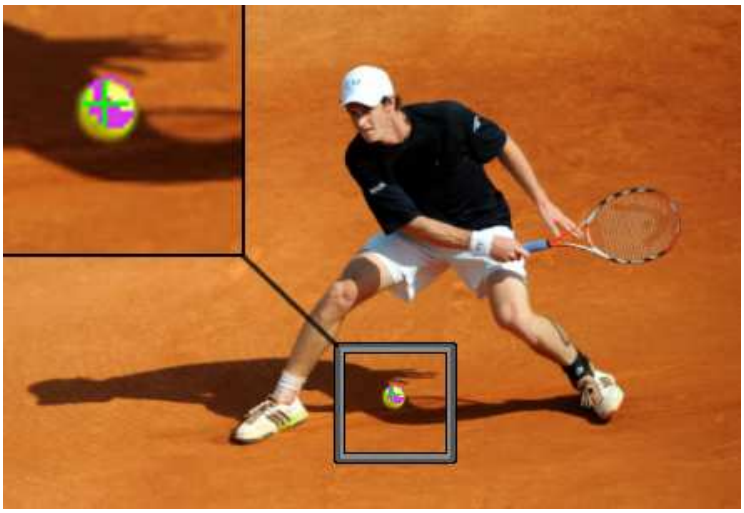
Le **premier pas** est d'**éliminer les composantes qui sont trop grandes ou trop petites**. Certaines composantes, en particulier si le décor est très uniforme, s'étendent sur toute l'image et n'indiquent certainement pas la position de la balle. D'autre, de même (on le voit sur l'image précédente), ne comportent qu'un ou deux pixels et ne pas valables. Il convient alors de filtrer ces composantes non valides. On notera h_c et w_c les hauteur et largeur moyenne de la balle cherchée, et Δ_c l'intervalle de hauteur et de largeur dans lequel on peut espérer la trouver. On obtient alors beaucoup moins de composantes (ici, 28 contre plusieurs milliers) :



Elimination par taille de la composante

La **deuxième étape**, pour terminer la détection, est **l'élimination des composantes par couleur**. Les composantes connexes restantes sont triées par rapport à la distance entre leur couleur moyenne et une couleur déterminée à l'avance. Initialement, je voulais choisir la couleur moyenne calculée au 1) sur les échantillons de balles, mais il est vite apparu qu'en pratique, la projection de la balle était bien plus proche d'une couleur comme $\gamma = (250, 250, 70)$, qui est celle que j'utilise actuellement, mais qu'il conviendrait de changer en cas d'altération des couleurs de l'image. Pour simplifier, je considère que la composante la plus proche est vraiement la balle.

La composante connexe la plus proche, ainsi que la position de la balle, est alors affichée :



4. Tests et performances

Pour une image de hauteur h et de largeur w, mon algorithme de détection de la balle opère donc comme suit :

- Elimination des des points qui n'ont pas une couleur valable et projection sur le plan jaune
  Complexité : $\mathcal{O}(hw)$, au pire, en nombre d'opérations de projection sur le plan jaune.
- Création de la grille de points
  Complexité : $\mathcal{O}(hw)$, au pire (lorsqu'il s'agit d'initialiser le regroupement avec chaque pixel de l'image
- Regroupement en composantes connexes
  Complexité : Chaque point n'est ajouté que dans une seule composante, la complexité est donc $\mathcal{O}(hw)$ en nombre d'ajouts d'un point dans une composante
- Elimination des composantes de taille non valable
  Complexité : Pour calculer la taille d'une composante connexe, il faut effectuer 4 opérations de comparaison par point lui appartenant, pour déterminer les bords supérieur, inférieur, gauche et droit ; chaque point appartenant à une unique composante, la complexité est au pire $\mathcal{O}(hw)$.
- Choix de la composante connexe représentant la balle
  Complexité : $\mathcal{O}(n)$ en nombre d'opération de comparaison, où n est le nombre de composantes connexes (la couleur moyenne d'une composante est inscrite lors du regroupement dans la structure choisie, pour préparer cette étape)
- Affichage de la composante trouvée

La complexité globale de l'algorithme est au pire $\mathcal{O}(hw)$.

Ses paramètres sont $d_{\max,y}$, $d_{\max,c}$, g_h, g_w, h_c, w_c, Δ_c, et γ.

Avec quelques tests, on voit que le but recherché a été atteint. Sur mes images de test, **la quasi-totalité des balles ont été détectées**, contrairement à mon implémentation de l'algorithme précédent, et les **performances lui sont similaires** : la plupart des images sont traitées en quelques secondes.

Cependant, **une intervention de l'utilisateur** plus importante est nécessaire : dans la transformée de Hough, le seul paramètre réellement important était la valeur minimale du gradient pour qu'un point soit un contour, dans la détection de contours. Ici, il est nécessaire d'ajuster les divers paramètres suivant le type de caméra, la taille de l'image attendue, la taille de la balle dans cette image, et le bruit extérieur.