Concours Commun INP (CCINP), France

Session 2018, filière PSI

Informatique

Durée : 3 heures

Ce que traite le sujet

- Parties I et II : présentation et modélisation de la trajectoire
- Partie III.1 et III.2 : fichiers et position 2D de la balle
- Partie III.3 à III.6 : position 3D, impact et reconstruction
- Partie IV : exploitation des données du Hawk-eye

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/ccinp/psi/2018/info

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

CONCOURS COMMUNS POLYTECHNIQUES

ÉPREUVE SPÉCIFIQUE - FILIÈRE PSI

INFORMATIQUE

Vendredi 4 mai : 8 h - 11 h

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites

Le sujet est composé de 4 parties, toutes indépendantes.

L'épreuve est à traiter en langage **Python**, sauf les questions sur les bases de données qui seront traitées en langage **SQL**. La syntaxe de Python est rappelée en annexe, page 16.
Les différents algorithmes doivent être rendus dans leur forme définitive sur la copie en respectant les éléments de syntaxe du langage (les brouillons ne sont pas acceptés).
Il est demandé au candidat de bien vouloir rédiger ses réponses **en précisant bien le numéro de la question traitée et, si possible, dans l'ordre des questions**. La réponse ne doit pas se cantonner à la rédaction de l'algorithme sans explication, les programmes doivent être expliqués et commentés.

Sujet : page 1 à page 15
Annexe : page 16

SYSTÈME D'AIDE À L'ARBITRAGE : LE HAWK-EYE

Partie I - Présentation générale du système

Le système Hawk-eye est un système d'aide à la décision arbitrale utilisé dans le sport de haut niveau, notamment le tennis, un des plus connus. Il a été développé par le Paul Hawkins en 1999 avec pour objectif d'augmenter la qualité des décisions arbitrales et fournir un résultat rapide, clair et précis lors des moments décisifs de rencontres sportives (**figure 1**). Il a été utilisé pour la première fois au tennis lors du Masters de Miami en 2006. Chaque joueur peut faire appel trois fois par set de cette décision.

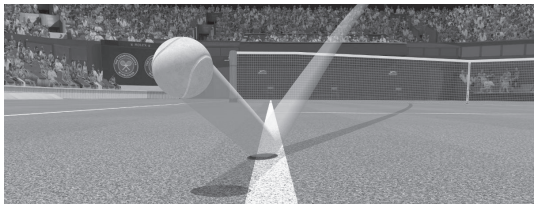


Figure 1 – Décision officielle du système Hawk-eye

Pour analyser la trajectoire et la position de la balle, le système Hawk-Eye se compose de dix caméras, réparties à égale distance autour du court de tennis dans les tribunes (**figure 2**). Ces caméras peuvent photographier "en rafale" des objets se déplaçant à une grande vitesse grâce à un capteur photographique. Elles peuvent enregistrer jusqu'à 1 000 images par seconde. Ces images sont ensuite envoyées au poste de commande du système.

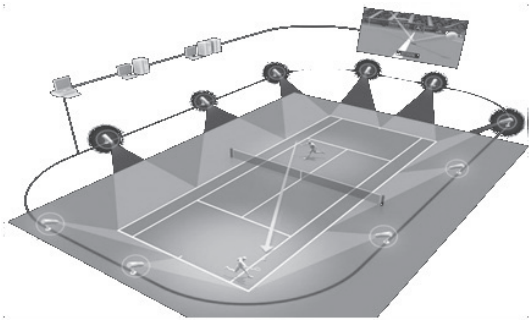


Figure 2 – Position des caméras haute vitesse autour du terrain de tennis

L'objectif de l'étude proposée est de réaliser le programme de suivi (tracking) de la trajectoire de la balle de tennis par le système Hawk-eye, la reconstruction de la trajectoire et enfin l'identification de la position de l'impact de la balle avec le sol pour savoir si la balle est dans les limites du terrain ou non. Afin de mieux appréhender le problème, nous commencerons par la modélisation et l'étude théorique de la trajectoire d'une balle de tennis et montrerons en quoi cette seule modélisation est insuffisante pour l'aide à l'arbitrage.

Dans tout le sujet, il sera supposé que les bibliothèques sont déjà importées dans le programme. Attention, l'utilisation des fonctions min et max ne sera pas acceptée.

Partie II - Modélisation de la trajectoire de la balle

L'objectif de cette partie est de déterminer la trajectoire théorique d'une balle de tennis et de montrer les limites du résultat obtenu.

La trajectoire d'une balle de tennis dépend de plusieurs paramètres :
- la vitesse de frappe par la raquette du joueur ;
- l'angle de frappe ;
- les frottements dans l'air ;
- la vitesse de rotation donnée à la balle par la raquette du joueur.

Le schéma de la **figure 3** représente un terrain de tennis pour une partie en simple et la trajectoire de la balle dans l'espace de coordonnées (x, y, z). Le repère est choisi de sorte que le plan $(0, \vec{x}, \vec{z})$ soit horizontal.

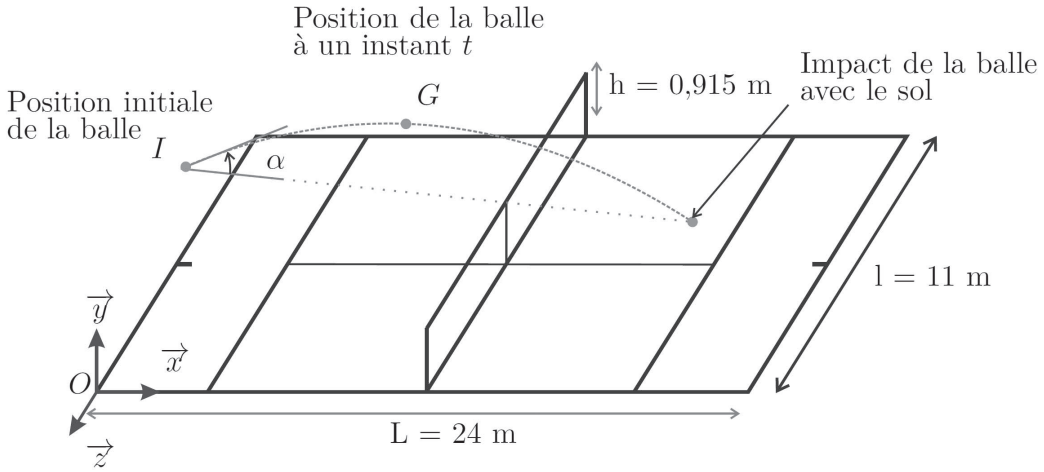


Figure 3 – Paramétrage du terrain de tennis et trajectoire d'une balle

Paramétrage
- La balle a une masse m et un rayon R.
- Le mouvement de la balle est étudié dans le référentiel $\mathcal{R}$ lié au terrain et supposé galiléen. On lui associe le repère $(O, \vec{x}, \vec{y}, \vec{z})$, avec O le point de coordonnées $(0, 0, 0)$.
- La position initiale de la balle est définie dans le référentiel supposé galiléen lié au terrain au point I par (x_0, y_0, z_0).
- La balle est repérée par la position de son centre d'inertie G auquel est associé le vecteur $\overrightarrow{OG} = (x + x_0).\vec{x} + (y + y_0).\vec{y} + (z + z_0).\vec{z}$.
- La vitesse de la balle est notée $\vec{V} = v_x\vec{x} + v_y\vec{y} + v_z\vec{z}$ et sa vitesse initiale $\vec{V_0}$.
- L'angle de frappe entre le plan $(O, \vec{x}, \vec{z})$ et $\vec{V_0}$ est α.
- La vitesse de rotation sur elle-même de la balle est notée $\vec{\Omega} = \Omega\vec{z}$.
- $\vec{g} = -g\vec{y}$ est l'accélération de la pesanteur.

Hypothèses
- On supposera **dans toute la suite de cette partie** que la balle **ne se déplace que dans le plan** $(O, \vec{x}, \vec{y})$. Par conséquent, à chaque instant, $v_z = 0$.
- Les différents efforts s'exerçant sur la balle de tennis sont représentés sur la **figure 4**, page 4. $\vec{t}$ est un vecteur unitaire tangent à la trajectoire et dirigé suivant le sens de la trajectoire. $\vec{n}$ est un vecteur unitaire normal à la trajectoire.

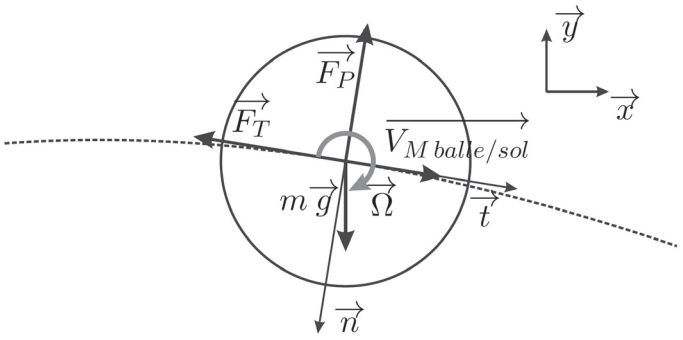


Figure 4 – Efforts de l'air sur la balle

Équation du mouvement de la balle
Le mouvement de la balle de tennis est défini par l'équation suivante

$$m\frac{d^2\overrightarrow{OG}}{dt^2} = m.\overrightarrow{g} + \overrightarrow{F_T} + \overrightarrow{F_P} \tag{1}$$

avec $\overrightarrow{F_T} = -\frac{1}{2}.\pi.R^2.\rho_{air}.C_1.V^2.\overrightarrow{t}$ et $\overrightarrow{F_P} = \rho_{air}.R^3.C_2.V.\Omega.\overrightarrow{n}$ où ρ_{air} est la masse volumique de l'air (environ 1 kg /m^3 à température ambiante), V la vitesse de la balle et C_1 et C_2 deux coefficients sans dimension qui dépendent du nombre de Reynolds.

L'équation (1) une fois projetée dans le plan $(O, \overrightarrow{x}, \overrightarrow{y})$ devient :

$$m\frac{d^2x(t)}{dt^2} = -\frac{1}{2}.\pi.R^2.\rho_{air}.C_1.V^2.\cos(\alpha) - \rho_{air}.R^3.C_2.V.\Omega.\sin(\alpha) \tag{2}$$

$$m\frac{d^2y(t)}{dt^2} = -m.g - \frac{1}{2}.\pi.R^2.\rho_{air}.C_1.V^2.\sin(\alpha) + \rho_{air}.R^3.C_2.V.\Omega.\cos(\alpha). \tag{3}$$

On pose $Y = \begin{bmatrix} u \\ v \\ x \\ y \end{bmatrix}$, avec $u(t) = \frac{dx(t)}{dt}$ et $v(t) = \frac{dy(t)}{dt}$ et dont la condition initiale est $Y = \begin{bmatrix} u_0 \\ v_0 \\ x_0 \\ y_0 \end{bmatrix}$.

Q1. Mettre les équations (2) et (3) sous la forme d'un problème de Cauchy du type : $\frac{\mathrm{d}Y}{\mathrm{d}t} = F(Y,t)$.

La résolution numérique des équations différentielles (2) et (3) repose sur leur discrétisation temporelle et conduit à déterminer à différents instants t_i une approximation de la solution.
On note u_i et v_i les approximations des composantes du vecteur vitesse $(u(t),v(t))$ et x_i et y_i les approximations des composantes du vecteur position $(x(t), y(t))$ à l'instant t_i. On note $\Delta t = t_{i+1} - t_i$, le pas de temps constant.

Le schéma d'Euler conduit à la relation de récurrence suivante :

$$Y_{i+1} = Y_i + \Delta t F(t_i, Y_i).$$

Q2. Compléter sur votre copie la fonction `euler(T,N,F,Y0)` ci-dessous permettant de construire le schéma d'Euler.

```
def euler(T,N,F,Y0) :
    Y=zeros((len(Y0),N))
    t=arange(0,T,T/N)
    #conditions initiales à compléter
    for i in range(1,N) :
        #zone à compléter
    return (t,Y)
```

La reconstruction de la trajectoire d'une balle de tennis à partir des lois physiques et des conditions initiales ne se révèle pas satisfaisante en terme de précision pour l'aide à la décision arbitrale. En effet, les paramètres climatiques (vent, pluie, pression atmosphérique...), de déformation de la balle ou tout autre aléa ne sont pas pris en compte *a posteriori*. Il convient donc d'adopter une autre méthode vérifiant à tout instant la position de la balle ; c'est ce que permet le système Hawk-eye. Nous allons nous intéresser dans la suite à une partie de l'algorithme permettant de vérifier si une balle est bonne et de reconstruire la trajectoire de la balle.

Partie III - Tracking et reconstruction de la trajectoire de la balle de tennis

Comme nous l'avons vu dans la partie présentation et plus particulièrement sur la **figure 2**, page 2, le système Hawk-eye est composé de 10 caméras réparties autour du terrain de tennis.
Pour pouvoir obtenir la trajectoire de la balle et déterminer le point d'impact de cette dernière avec le terrain, il est nécessaire de faire appel à une unité de traitement des données de différentes caméras.
Cette unité de traitement des données fait appel à un algorithme de calcul qui se décompose en 7 étapes :

- Étape ① : on détermine les limites du terrain ;
- Étape ② : on réalise le calibrage de la caméra (détermination de sa position, son orientation et ses paramètres intrinsèques) ;
- Étape ③ : on identifie les pixels représentant la balle et on calcule la position 2D dans chaque image de chaque caméra ;
- Étape ④ : on calcule la position 3D de la balle par triangularisation ;
- Étape ⑤ : on assemble les images obtenues ;
- Étape ⑥ : on détermine l'impact de la balle avec le sol sur le terrain ;
- Étape ⑦ : on reconstruit la trajectoire 3D de la balle que l'on affichera pour les spectateurs.

Nous nous proposons dans la suite du sujet de nous intéresser plus spécifiquement aux étapes ③, ④, ⑤, ⑥ et ⑦.

III.1 - Détermination de la taille des fichiers récupérés

L'image acquise par la caméra est une image en couleur constituée de trois couches de pixels (Red - Green - Blue). La résolution de l'écran est de 1 024 pixels horizontalement et 768 verticalement. Les données de l'image sont stockées dans un tableau de type `list` à trois dimensions : la première dimension correspond à la coordonnée selon x, la seconde à la coordonnée selon y et la troisième correspond à la couleur (rouge, vert ou bleu, indice 0, 1 ou 2).

Ainsi, les dimensions du tableau sont : $n \times m \times 3$ où $n = 1\,024$ et $m = 768$. La valeur associée à chaque pixel est un entier compris entre 0 et 255.

Un point joué désigne l'ensemble des coups exécutés par les joueurs, service compris, jusqu'à ce que l'un d'entre eux commette une faute ou un coup gagnant. Un échange désigne un couple de coups joués par les joueurs.

Q3. On suppose que l'on stocke dans la variable `image1` une image enregistrée par la caméra. Indiquer la valeur des expressions suivantes :

- `len(image1)`
- `len(image1[12][244])`

Indiquer le type de l'expression suivante :

- `len(image1[12][244][2])`

Q4. Déterminer, en justifiant succinctement votre calcul, la taille de l'espace mémoire occupé par le tableau représentant l'image émise par la caméra.

Q5. Estimer un ordre de grandeur de l'espace de stockage à prévoir pour sauvegarder toutes les images des 10 caméras rapides (1 000 images/s) correspondant à un point joué, puis à un set. Un set comporte en moyenne 100 points joués. On pourra estimer à 10 s la durée moyenne d'un point joué. Indiquer quel moyen de stockage pourra convenir à la sauvegarde des données.

Q6. Définir ce qu'on appelle la mémoire vive d'un ordinateur. Indiquer les inconvénients liés à l'utilisation des données décrites à la question précédente.

III.2 - Calcul de la position 2D de la balle (étape ③)

L'objectif de l'algorithme que nous allons étudier ici est d'identifier la position de la balle en prenant en compte sa taille et sa forme sur chaque image enregistrée à l'aide d'une caméra rapide.

L'algorithme doit renvoyer les coordonnées x et y de la balle dans le repère $(O, \vec{x}, \vec{y}, \vec{z})$ et l'instant correspondant t lorsque celle-ci est détectée. Si la balle n'est pas détectée dans le champ de vision de la caméra, l'information retournée sera `None`.

L'algorithme proposé se déroule en plusieurs étapes :

- première étape : détecter tous les objets en mouvement grâce à l'analyse de deux images successives,
- deuxième étape : éliminer les objets en mouvement qui ne peuvent pas correspondre à une balle de tennis selon des critères géométriques,
- troisième étape : éliminer les "balles fausses" détectées en prenant en compte la trajectoire de la balle.

La première étape est réalisée en comparant deux images successives (donc séparées d'un laps de temps très bref). Ces 2 images (`image1` et `image2`) sont quasiment identiques exceptées les zones en mouvement. L'objectif est de créer un nouveau tableau de dimension $n \times m$ (avec $n = 1\,024$ et $m = 768$) dans lequel l'élément correspondant à un pixel est un entier compris entre 0 et 255. Cet entier correspond à un niveau de gris (le noir correspondant à l'entier 0 et le blanc à l'entier 255). Les zones blanches ou claires seront considérées comme des objets en mouvement.

Les images sont celles décrites en début de sous-partie **III.2** et correspondent à des tableaux à 3 dimensions de type `list` : $n \times m \times 3$. Pour détecter les zones en mouvement, l'algorithme propose de

calculer pour chaque pixel la somme des différences au carré associées à chaque couleur

$$\Delta = (r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2 \quad (4)$$

où r_i, g_i et b_i représentent respectivement les niveaux de rouge, vert et bleu d'un pixel de coordonnées (x, y) de l'`image1` et de l'`image2`.
En cas de variation importante de couleur, Δ peut atteindre une valeur supérieure à 255 ce qui pose des difficultés dans la suite de l'algorithme. On veut garantir $0 \leq \Delta \leq 255$ et on impose donc la valeur 255 dans le cas où la formule précédente fournirait une valeur supérieure.

Q7. Écrire une fonction `detection(image1,image2)` qui prend en argument 2 images décrites à la sous-partie **III.2** (page 5) et qui renvoie un tableau de dimension $n \times m$ d'entiers compris entre 0 et 255 dans lequel chaque élément correspond à Δ. Écrire l'instruction qui permet d'affecter à la variable `image_gray` le résultat de cette fonction appliquée à deux images `im1` et `im2`.

Le tableau ainsi obtenu est ensuite filtré pour éliminer les pixels trop clairs qui correspondent à des parties figées de l'image. Le seuil doit pouvoir être ajusté pour chaque caméra suivant la luminosité et les conditions météorologiques.

Q8. Écrire une fonction `filtre(image,seuil)` qui prend en argument un tableau à deux dimensions (de type `list` de `list`) et qui renvoie un nouveau tableau de même dimension que celui passé en argument. Chaque élément de ce nouveau tableau prend la valeur 0 si l'élément correspondant du tableau `image` est inférieur à la valeur `seuil` et prend la valeur 1 sinon.

Nous avons alors obtenu un tableau qui permet de repérer tous les éléments en mouvement dans l'image. Il reste maintenant à détecter les éléments en mouvement qui peuvent être une balle de tennis.
Compte-tenu de la position des caméras latérales et de leur résolution, une balle de tennis occupe au minimum 4 pixels.
Nous allons commencer par éliminer tous les pixels en mouvement qui sont isolés dans le tableau : aucun des pixels adjacents parmi les 8 possibles n'est considéré en mouvement.

Q9. La fonction `filtre_pixel(image)` prend en argument un tableau `image` à deux dimensions, rempli de 0 et de 1. Cette fonction doit permettre de balayer tous les éléments du tableau `image` et de tester s'ils correspondent à un pixel isolé.
Proposer deux schémas permettant d'illustrer le cas d'un pixel isolé et donc à éliminer et celui d'un pixel à conserver.
On considère un pixel de coordonnées (p,c) tel que ce pixel ne soit pas situé sur le bord de l'image. Parmi les 4 tests suivants, choisir celui qui permet de détecter si ce pixel est isolé et, si c'est le cas, de l'éliminer.

Test 1

```
if image[c-1][p-1]==0 or image[c-1][p+1]==0 or image[c+1][p-1]==0
   or image[c+1][p+1] == 0:
    image[p][c]=0
```

Test 2

```
if image[c-1][p-1]==0 and image[c-1][p+1]==0 and image[c+1][p-1]==0
    and image[c+1][p+1] == 0:
    image[p][c]=0
```

Test 3

```
if image[p][c]==1:
    nb_pixel=-1
    for k in [p-1,p,p+1]:
        for j in [c-1,c,c+1]:
            nb_pixel += image[k][j]
    if nb_pixel==0:
        image[p][c]=0
```

Test 4

```
if image[p][c]==1:
    nb_pixel=-1
    for k in range(p-1,p+1):
        for j in range(c-1,c+1):
            nb_pixel += image[k][j]
    if nb_pixel==0:
        image[p][c]=0
```

Une fonction est ensuite implémentée pour éliminer les objets en mouvement trop grands et qui ne peuvent correspondre à une balle de tennis. Ces objets sont en général représentés par des chaînes de pixels à 1 représentant le contour d'un objet en mouvement. Cette fonction ne sera pas étudiée ici.

Le script présenté ci-dessous permet de réaliser le prétraitement à partir des images extraites des vidéos enregistrées par une caméra. Les images issues d'un même point sont stockées dans un dossier `sequence_####` où `####` correspond au numéro du point joué. Ces images sont enregistrées au format `.bmp` et sont nommées `image_#####.bmp` où `#####` est le numéro de l'image dans la séquence vidéo.

```
1       point = input("Rentrer le numéro du point litigieux à
    analyser")
2       dirs = listdir("sequence_" + point)
3       for image in dirs:
4           index_max = 0
5           index = int(image[6:11])
6           if index >= index_max:
7               index_max = index
8
9       seq_balle =[[x0,y0]]# [x0,y0] position initiale de la balle
10                          # donnée par un autre traitement
11      for image in dirs:
12          index_init = index_max - 3000 # indice de la première
    image de la séquence analysée
13          if int(image[6:11])== index_init :
14              image2=imread("sequence_"+ point +"/"+image)
15          elif int(image[6:11])> index_init :
16              image1 = deepcopy(image2)
17              image2=imread("sequence_"+ point +"/"+image)
18              liste_balles = traitement(image1,image2)
19          seq_balle.append(liste_balles)
```

Éléments de documentation

- la commande `listdir` s'applique à un dossier et renvoie une liste de tous les noms de fichiers contenus dans ce dossier,
- la commande `imread` construit un tableau $n \times m \times 3$ à partir d'une image bitmap (`.bmp`),
- la fonction `traitement` renvoie une liste des positions des balles éventuelles détectées dans une image. Cette fonction fait appel aux fonctions précédemment programmées. (Il n'est pas demandé de détailler cette fonction).

Q10. Commenter les lignes 2 et 3, puis indiquer l'intérêt de chaque boucle `for`. Préciser l'intérêt de ce programme.

Après les différents traitements définis précédemment, nous avons pour chaque image une liste des coordonnées de la balle. Malgré ces traitements, une image peut contenir plusieurs points pouvant être assimilés à la balle de tennis. Par exemple, dans le cas d'une image i qui contient 2 balles possibles, on stocke les coordonnées dans une liste :

$$\text{image } i : \left[x_{i1}, y_{i1}, x_{i2}, y_{i2}\right].$$

Chaque liste associée à une image est placée dans une liste qui regroupe ainsi toutes les images de la séquence à analyser :

$$\texttt{seq_balle} = \left[\left[x_0, y_0\right], ..., \underbrace{\left[x_{i1}, y_{i1}, x_{i2}, y_{i2}\right]}_{\text{image } i}, \underbrace{\left[x_{(i+1)1}, y_{(i+1)1}, x_{(i+1)2}, y_{(i+1)2}\right]}_{\text{image } i+1}, ...\right]$$

À partir de la liste `seq_balle` et de la vitesse initiale, nous allons chercher à ne conserver que les "balles bonnes" en prenant en compte la trajectoire. On suppose qu'entre deux images successives, la vitesse de la balle varie peu. Ainsi, à partir de la balle détectée à l'image $i-1$ (de coordonnées (x_{i-1}, y_{i-1})), nous allons chercher la balle suivante dans une zone rectangulaire dont le centre (x_c, y_c) sera calculé à partir de la position de la balle (x_{i-1}, y_{i-1}) et du vecteur déplacement $(d_{x\,i-1}, d_{y\,i-1})$:

$$x_c = x_{i-1} + d_{x\,i-1} \qquad \text{et} \qquad y_c = y_{i-1} + d_{y\,i-1}.$$

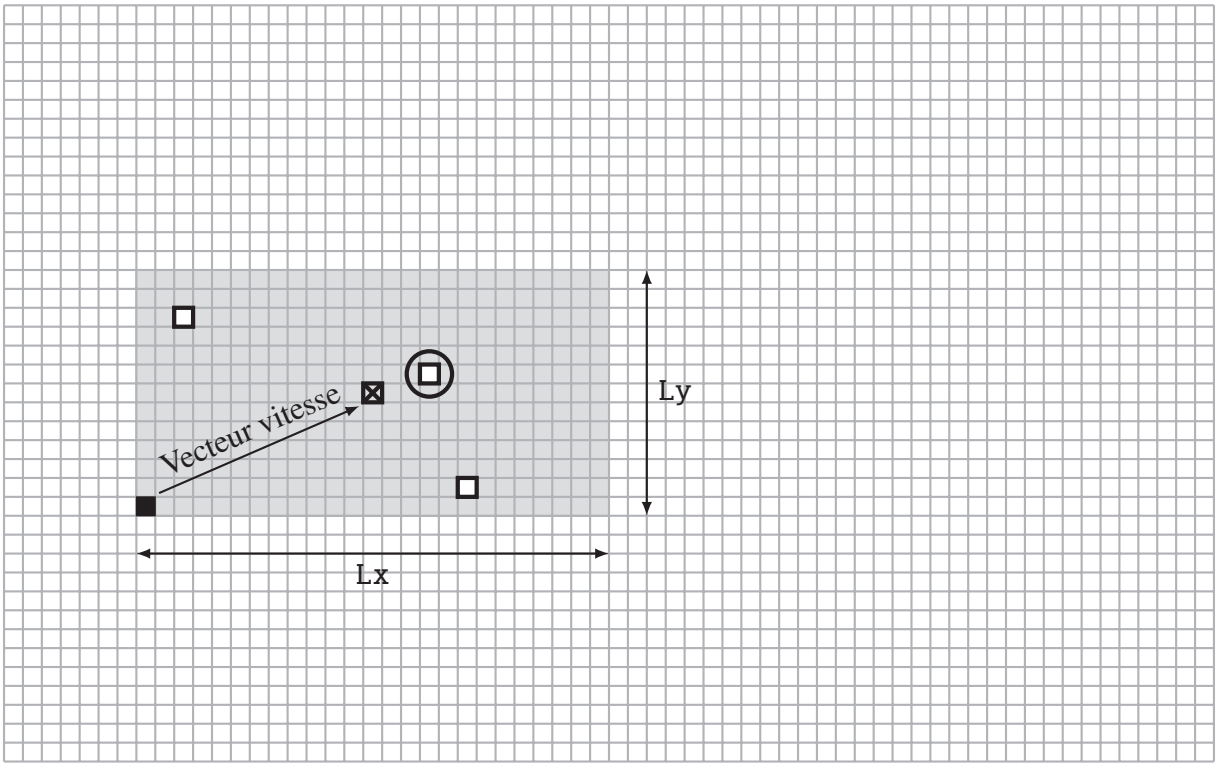
Le rectangle aura pour dimension $\texttt{Lx} = 2d_{x\,i-1} + 1$ suivant x et $\texttt{Ly} = 2d_{y\,i-1} + 1$ suivant y avec au minimum 3 pixels suivant x et 3 pixels suivant y, c'est-à-dire au minimum $\texttt{Lx} = 3$ et $\texttt{Ly} = 3$.

- Si plusieurs balles sont présentes dans cette zone (balles candidates), on choisira celle dont la position sera la plus proche du centre. En cas d'égalité, on conservera arbitrairement l'une des 2 positions.
- Si aucune balle n'est présente dans la zone, on renverra pour l'image $i+1$ la liste `[None, None]` et on cherchera directement une balle dans l'image suivante $i+2$ en doublant les dimensions de la zone de recherche $2\texttt{Lx} - 1$ et $2\texttt{Ly} - 1$ et en prenant comme pixel central : $x_c = x_i + 2d_{x\,i-1}$ et $y_c = y_i + 2d_{y\,i-1}$.

Les images successives étant prises à intervalles de temps égaux, on définit le vecteur déplacement à l'instant i (ou à l'image i) comme une liste à deux éléments :

$$\texttt{dep_i} = \left[d_{x\,i}, d_{y\,i}\right] = \left[x_i - x_{i-1}, y_i - y_{i-1}\right].$$

L'algorithme proposé dans cette sous-partie est illustré **figure 5**, page 10.



■ Balle détectée à l'image précédente
⊠ Centre de la zone de recherche
□ Balles candidates
◎ Balle détectée

Figure 5 – Détection des "balles bonnes"

Q11.

a) Écrire une fonction `deplacement(pos1,pos2)` qui prend en argument 2 listes à 2 éléments (`pos1` et `pos2`) et qui renvoie le vecteur déplacement associé (sous la forme d'une liste à deux éléments). On précisera le type des éléments constituants les listes.
Écrire la ou les instructions permettant d'affecter aux variables `Lx` et `Ly` les dimensions de la zone de recherche.

b) Écrire une fonction `distance_quad(xc,yc,liste_balle_i)` où `xc,yc` sont les coordonnées du pixel central et `liste_balle_i` la liste des balles possibles. Cette fonction doit renvoyer une liste des carrés des distances séparant chaque balle possible du pixel central.

c) Écrire une fonction `cherche_balle(xc,yc,Lx,Ly,liste_balle_i)` où `xc,yc` sont les coordonnées du pixel central, `Lx,Ly` les dimensions de la zone de recherche et `liste_balle_i` la liste des balles possibles. Cette fonction doit renvoyer une liste contenant les coordonnées de la balle détectée ou la liste `[None, None]` si aucune balle n'est détectée. Cette fonction fera appel à la fonction `distance_quad`.

d) Écrire une fonction `traj_balle` qui prend en argument la liste `seq_balle` définie précédemment et la vitesse initiale `vit_init` (liste à 2 éléments). Cette fonction doit renvoyer la liste des "balles bonnes" présentes dans la zone de recherche :

$$[[x_0, y_0], [x_1, y_1], ..., [x_i, y_i], ...].$$

On ne traitera pas le cas où dans 2 images successives aucune balle ne se trouve dans la zone de recherche.

Q12. Indiquer les limites ou défauts de l'algorithme étudié dans cette sous-partie.

III.3 - Calcul de la position 3D de la balle par triangularisation (étape ④)

L'objectif de cette sous-partie est de déterminer la position de la balle dans le repère global défini par rapport au terrain à partir de la position de la balle dans deux images de deux caméras différentes.

Une fois que la configuration des caméras a été calibrée, la position de la balle peut être reconstruite dans le repère 3D par stéréo-triangulation. Cela consiste à déterminer la localisation du centre de la balle (noté M de coordonnées (x, y, z) dans le repère global) à partir de la connaissance de la position du centre de la balle dans les images obtenues par deux caméras. La position de la balle dépend de l'espacement e des caméras, également appelé "longueur de la ligne de base stéréo des caméras", de la distance focale f des caméras et de la disparité d qui est la distance entre les centres de la balle dans les images de chaque caméra, ainsi que des coordonnées du centre de la balle dans chacune des images (**figure 6**).

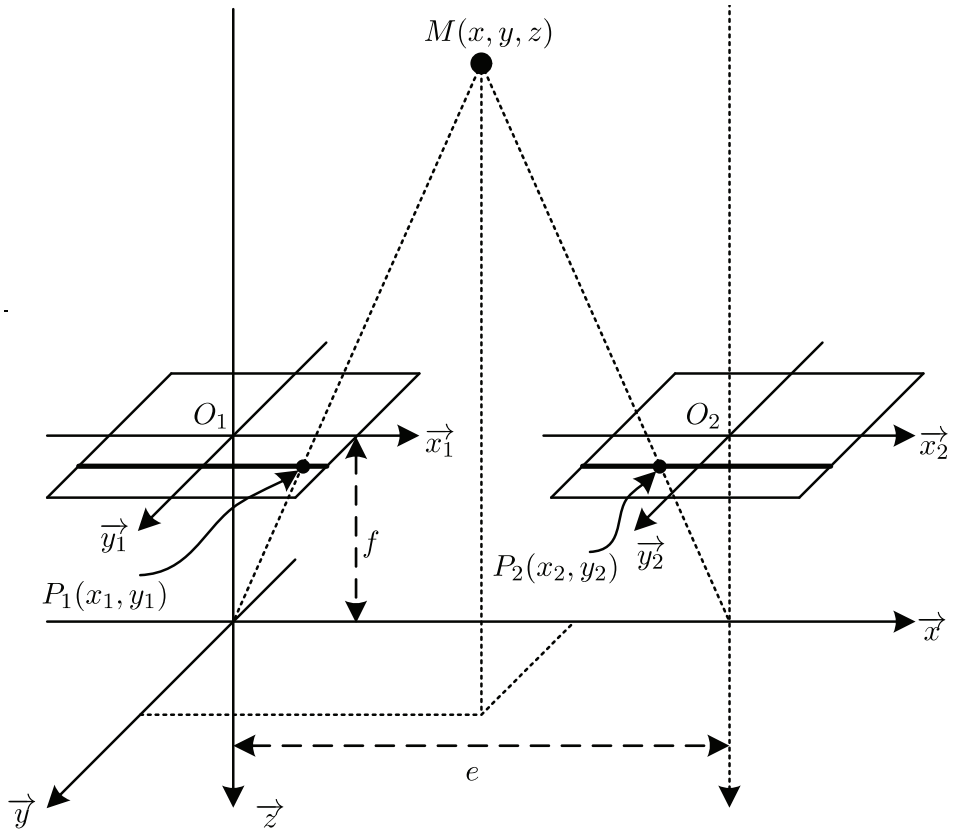


Figure 6 – Position de la balle à l'instant i dans les images des caméras 1 et 2

Hypothèses

- On s'intéressera ici à la triangularisation sur deux caméras : les caméras 1 et 2.
- On suppose que la balle est bien détectée par les deux caméras et que les coordonnées relevées correspondent au centre de la balle. Les coordonnées du centre de la balle à l'instant i sont notées respectivement : P_1 de coordonnées (x_1, y_1) pour la caméra 1 et P_2 de coordonnées (x_2, y_2) pour la caméra 2 dans leurs repères respectifs.
- On suppose que les deux caméras sont disposées à la même hauteur autour du terrain et que leur plan focal est identique.
- On suppose que le temps est bien synchronisé sur les deux caméras.
- On suppose que tous les paramètres de calibration des deux caméras sont connus (matrices intrinsèques et extrinsèques, distance focale, position...).

On calcule les coordonnées du centre de la balle à l'instant i dans le repère de la caméra 1 $(x1i, y1i, z1i)$ grâce aux formules suivantes : $x1i = \frac{e}{d}x_1$, $y1i = \frac{e}{d}y_1$ et $z1i = \frac{e}{d}f$, avec $d = x_1 - x_2$. Ces coordonnées sont placées dans une liste telle que : `pos1i=[x1i,y1i,z1i]`.

Q13. Écrire une fonction `pos_loc(e,f,x1,x2,y1,y2)` qui prend en argument les positions de la balle dans la caméra 1 ($x1$,$y1$) et la caméra 2 ($x2$,$y2$) à un instant i puis calcule et renvoie une liste des coordonnées $(x1i, y1i, z1i)$ de la balle dans le repère local de la caméra 1 à un instant i `pos1i=[x1i,y1i,z1i]`.

On calcule ensuite les coordonnées x, y et z dans le repère global à l'aide des matrices de passage de la caméra 1 par rapport au repère global. La position et l'orientation de la caméra 1 par rapport au repère global du système sont définies par trois matrices : T sa matrice de translation suivant $\vec{x}$, $\vec{y}$ et $\vec{z}$, R_x sa matrice de rotation d'angle θ suivant $\vec{x}$ et R_y sa matrice de rotation d'angle γ suivant $\vec{y}$. On obtient la même chose pour la caméra 2 en remplaçant les 1 par des 2. Ces matrices sont supposées connues à chaque instant (résultats de l'étape ② non abordée). La **figure 7** illustre la position de la caméra 1 à l'instant t.

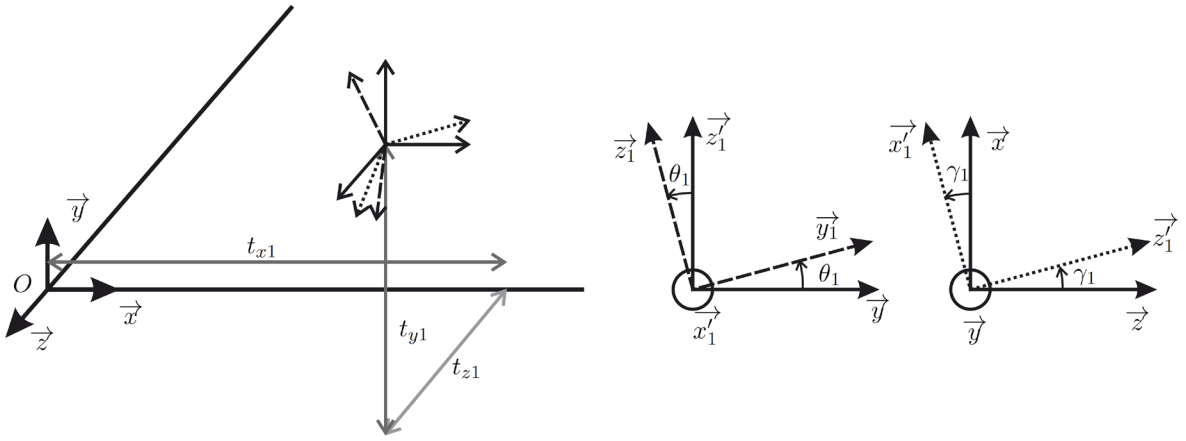


Figure 7 – Position de la caméra 1 à l'instant t

$$T_1 = \begin{pmatrix} t_{x1} \\ t_{y1} \\ t_{z1} \end{pmatrix}; R_{x1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 \\ 0 & \sin\theta_1 & \cos\theta_1 \end{pmatrix} \text{et } R_{y1} = \begin{pmatrix} \cos\gamma_1 & 0 & \sin\gamma_1 \\ 0 & 1 & 0 \\ -\sin\gamma_1 & 0 & \cos\gamma_1 \end{pmatrix}.$$

On rappelle que si R_1 est la matrice de rotation entre les bases 0 et 1, alors $R_1.U_1 = U_0$, avec U_0 un

vecteur défini dans la base 0 et U_1 un vecteur défini dans la base 1.

Q14. Écrire une fonction `pos_glo(pos1i,T1,Rx1,Ry1)` qui prend en argument la liste `pos1i` et les matrices de translation et rotations `T1`, `Rx1` et `Ry1`, qui calcule et renvoie une liste des coordonnées de la balle dans le repère global à l'instant i `posgi=[xgi,ygi,zgi]`. On pourra utiliser les fonctions de numpy pour effectuer les produits matrice-vecteur qui sont rappelées en annexe.

III.4 - Assemblage des positions 3D (étape ⑤)

L'objectif de cette partie est d'assembler les positions 3D de la balle dans le repère global obtenues à l'étape ④.

Les hypothèses sont les mêmes que dans la sous-partie **III.3**. Les coordonnées locales successives de la balle dans le repère de la caméra 1 sont stockées dans une liste `coord_loc=[[x10,y10,z10], [x11,y11,z11],...,[x1i,y1i,z1i],...,[x1N,y1N,z1N]]`. La trajectoire complète est obtenue en assemblant les positions successives de la balle dans la repère global.

Q15. Écrire une fonction `traj3D(coord_loc,T1,Rx1,Ry1)` qui prend en argument la liste `coord_loc` et renvoie une liste de l'assemblage dans l'ordre chronologique des N coordonnées de la balle dans le repère global `coord_glo=[[xg0,yg0,zg0],[xg1,yg1,zg1],..., [xgi,ygi,zgi],...,[xgN,ygN,zgN]]`. On pourra utiliser des fonctions de la sous-partie **III.3**.

III.5 - Détermination de l'impact de la balle avec le sol sur le terrain (étape ⑥)

L'objectif de cette partie est de déterminer le point d'impact de la balle avec le sol afin de savoir si la balle est restée dans les limites du terrain ou non.

Les hypothèses sont les mêmes que dans la sous-partie **III.3**.
Cette partie de l'algorithme est primordiale, en effet, c'est à partir du résultat obtenu par cette partie de l'algorithme que l'arbitre rendra sa décision. Le système réel permet de déterminer la forme de l'impact entre la balle et le sol en fonction de la déformation de la balle. Afin de faciliter la résolution du problème, la déformation de la balle ne sera pas prise en compte.

Hypothèses

- On suppose que la balle est indéformable (et donc que le point d'impact est ponctuel) et de rayon 0,033 m.
- La position du terrain (et donc les lignes) est parfaitement connue dans le repère global.
- On considère dans notre cas que le point étudié se fait durant l'échange de balle entre les joueurs et non sur un service.
- On suppose que le terrain est parfaitement plat.

Les coordonnées du point d'impact de la balle avec le sol sont déterminées de la manière suivante : on vient détecter quand la coordonnée suivant y est égale à R et on relève les positions correspondantes suivant x et z.
Si la position y comprise entre R et $R+\varepsilon$ ne se trouve dans aucune image, on détermine les coordonnées moyennes suivant x et z des deux images successives où il y a eu inversion du sens de déplacement de la balle selon y (descente, puis remontée). La donnée ε sera notée `eps` dans le programme.

Q16. Écrire une fonction `det_impact(coord_glo,eps)` qui prend en argument la liste `coord_glo` et le flottant `eps` puis calcule et renvoie la position de l'impact de la balle dans le repère global : une liste de dimension 2 `impact=[x_imp,z_imp]`.

Le terrain comme présenté sur la **figure 3** fait $L = 24$ m de long sur $l = 11$ m de large. On considère que le point origine du repère global est le point O.

Q17. Écrire une fonction `res_final(impact,L,l)` qui prend en argument la liste `impact`, les dimensions du terrain `L` et `l` et qui renvoie IN si l'impact de la balle a eu lieu dans les limites du terrain et OUT sinon.

III.6 - Reconstruction de la trajectoire 3D de la balle (étape ⑦)

L'objectif de cette sous-partie est d'afficher la trajectoire 3D de la balle reconstruite à l'étape ⑤ pour que les spectateurs puissent la visualiser.

Q18. À partir de la documentation sur le tracé 3D en Python, fournie en annexe, écrire une fonction `vis_traj3D(coord_glo)` qui prend en argument la liste `coord_glo` et permet d'afficher la trajectoire 3D de la balle.

Partie IV - Exploitation des données fournies par le Hawk-eye

L'objectif de cette partie est de montrer comment utiliser les informations fournies par le système Hawk-eye à des fins d'entraînement ou de visualisation à la télévision.

Le système Hawk-eye est une mine d'informations pour analyser les statistiques des matchs de tennis. Les données sont ainsi utilisées par les chaînes de télévisions lors des retransmissions des matchs mais également par les entraîneurs après les matchs en vue d'adapter leurs programmes d'entraînement.

Ces données d'un tournoi sont stockées dans une base de données constituée de deux tables.
La table MATCHS contient les différents matchs d'un tournoi avec les attributs :

- `id` : identifiant de type entier, clé primaire ;
- `nom` : nom du match de type texte ;
- `numero` : numéro du match dans la planification du tournoi ;
- `date` : date où le match s'est déroulé ;
- `joueur1` : nom du premier joueur ;
- `joueur2` : nom du deuxième joueur (les joueurs 1 et 2 sont rangés par ordre alphabétique) ;
- autres attributs non détaillés...

La table POINTS contient des entités correspondant aux points joués lors d'un match. Elle contient les attributs :

- `id` : identifiant de type entier, clé primaire ;
- `mid` : identifiant du match correspondant à la définition de type entier ;
- `nombre` : nombre d'échanges de type entier ;
- `fichier` : nom du fichier image de la trajectoire (stockée) correspondant au point ;
- autres attributs non détaillés...

Q19. Rappeler la définition et l'intérêt d'une clé primaire.

Q20. Écrire une requête SQL permettant d'afficher les identifiants des matchs joués par Federer, joueur pris pour exemple.

Q21. Écrire une requête SQL permettant d'afficher le nombre d'échanges maximum lors du match dont l'identifiant du match est `mid=4`.

Q22. Écrire une requête SQL permettant de récupérer le nom des fichiers de toutes les images qui correspondent au match dont le nom est "Federer-Murray".

FIN

ANNEXE

Rappels des syntaxes en Python

Remarque : sous Python, l'import du module numpy permet de réaliser des opérations pratiques sur les tableaux : from numpy import *. Les indices de ces tableaux commencent à 0.

	Python
tableau à une dimension	`L=[1,2,3]` (liste) `v=array([1,2,3])` (vecteur)
accéder à un élément	`v[0]` renvoie 1
ajouter un élément	`L.append(5)` uniquement sur les listes
tableau à deux dimensions (matrice) :	`M=array(([1,2,3],[3,4,5]))`
accéder à un élément	`M[1,2]` ou `M[1][2]` donne 5
extraire une portion de tableau (2 premières colonnes)	`M[:,0:2]`
tableau de 0 (2 lignes, 3 colonnes)	`zeros((2,3))`
dimension d'un tableau `T` de taille (i, j)	`T.shape` donne `[i,j]`

produit matrice-vecteur (est identique pour le produit matrice-matrice)	`a = array`
	`([[1,2,3],[4,5,6],[7,8,9]])`
	`b = array([1,2,3])`
	`print (a.dot(b))`
	`>>> array([14,32,50])`

séquence équirépartie quelconque de 0 à 10.1 (exclus) par pas de 0.1	`arange(0,10.1,0.1)`
définir une chaîne de caractères	`mot="Python"`
taille d'une chaîne	`len(mot)`
extraire des caractères	`mot[2:7]`
boucle For	`for i in range(10):` `  print(i)`

condition If	`if (i>3):`
	`  print(i)`
	`else:`
	`  print("hello")`

définir une fonction qui possède un argument et renvoie 2 résultats	`def f(param):`
	`  a=param`
	`  b=param*param`
	`  return a,b`

tracé d'une courbe de deux listes de points x et y	`plot(x,y)`
tracé d'une courbe de trois listes de points x, y et z	`gca(projection='3d').plot(x,y,z)`
ajout d'un titre sur les axes d'une figure	`xlabel(texte)` `ylabel(texte)`
ajout d'un titre principal sur une figure	`title(texte)`

IMPRIMERIE NATIONALE – 18 1065 – D'après documents fournis

Travailler les chapitres de ce sujet

Prepa Help suit le programme de PSI chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Bases de Python : expressions, types, instructions, fonctions
- Équations différentielles linéaires du second ordre
- Images : matrices de pixels, opérations, convolution
- Architecture de l'ordinateur et représentation des nombres
- Listes et algorithmes de recherche (séquentielle, dichotomique)

Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/ccinp/psi