CORRIGÉ [officiel → plein de fautes...]

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

$$\frac{du(t)}{dt} = \frac{1}{m}(-\frac{1}{2}.\pi.R^2.\rho_{air}.C_1.V.u + \rho_{air}.R^3.C_2.\Omega.v)$$
$$\frac{d^2y(t)}{dt^2} = \frac{1}{m}(-m.g - \frac{1}{2}.\pi.R^2.\rho_{air}.C_1.V.v + \rho_{air}.R^3.C_2.\Omega.u)$$
$$\frac{dx(t)}{dt} = u(t)$$
$$\frac{dy(t)}{dt} = v(t)$$

1pt/ligne.

On en déduit :

$$F(Y,t) = F\left(\begin{bmatrix} u \\ v \\ x \\ y \end{bmatrix}, t\right) = \begin{bmatrix} \frac{1}{m}(-\frac{1}{2}.\pi.R^2.\rho_{air}.C_1.V.u + \rho_{air}.R^3.C_2.\Omega.v) \\ \frac{1}{m}(-m.g - \frac{1}{2}.\pi.R^2.\rho_{air}.C_1.V.v + \rho_{air}.R^3.C_2.\Omega.u) \\ u(t) \\ v(t) \end{bmatrix}$$

avec $V = \sqrt{u^2 + v^2}$

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)
    Y[0] = Y0
    for i in range(1,N) :
        Y[i] = (F(Y[i-1], t[i-1]))*T/N + Y[i-1]
    return (t,Y)
```

pratique Y[:,i] en numpy

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 :

- `image1[12][244][2]`

- `len(image1)` = 1024 (ou 768 suivant la compréhension)
- `len(image1[12][244])` = 3
- `image1[12][244][2]` type entier / integer / uint8 (ou erreur si on laisse le len)

wish U well

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

Chaque pixel est représenté par les 3 couleurs qui sont codées sur 8 bits, c'est-à-dire sur un octet. $M = 1024 \times 768 \times 3 = 2359296$ o. On peut remarquer que $1024 = 2^{10}$. Ainsi on obtient le résultat en kio simplement : $M = 768 \times 3 = 2304$ kio. En Mio, en divisant à nouveau par 1024 : $M = 2{,}25$ Mio. On obtient donc un ordre de grandeur habituel. On ne fera pas forcément la disctinction kio et ko.

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.

Les candidats doivent indiquer un ordre de grandeur :

pour un point et en Mio : $10 \times 1000 \times 10 \times 2{,}25$

Durée moyenne	en octets	en kio	en Mio	en Gio
pour un point : $\Delta t = 10$ s	235929600000	230400000	225000	219,7
pour un set (100 points)	$2{,}37.10^{13}$	$2{,}30.10^{10}$	$2{,}25.10^{7}$	21970

Les données pourront être stockées sur un disque dur (mémoire de masse).

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.

La mémoire vive (ou RAM : random access memory) fait partie de la mémoire principale. Elle est accessible très rapidement par le processeur. Elle permet de stocker temporairement les données et programmes en cours de traitement. Elle est volatile et rapide d'accès. Ordre de grandeur : 4 à 16 Go.
Les fichiers sont de taille considérable. Pour que le système soit performant (réponse rapide) il faut que les données correspondant au dernier échange soit stockées dans la RAM pour être traitées rapidement...

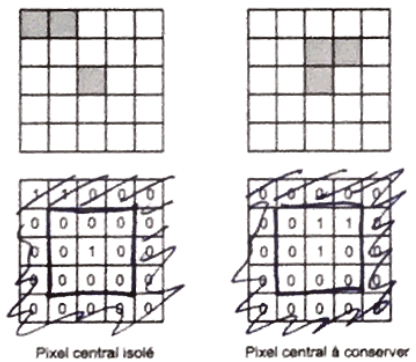
Q7. Écrire une fonction `detection(image1,image2)` qui prend en argument 2 images décrites à la sous-partie **III.2** (page **??**) et qui renvoie un tableau de dimension $n \times m$ d'entiers compris entre 0 et 255 dans lequel chaque élément correspond à Δ. Écrire l'instruction qui permet d'affecter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

```
def detection(image1,image2):
    """prend en arguments 2 tableaux à 3 dimensions
    renvoie un tableau à 2 dimensions """
    tab = []
    nb_lig = len(image1)
    nb_col = len(image1[0])
    for k in range(nb_lig):
        tab.append([])
        for j in range(nb_col):
            delta = 0
            for c in range(3):
                delta += (image1[k][j][c] - image2[k][j][c] )**2

            if delta > 255:
                delta = 255
            tab[k].append(delta)
    return tab

image_grey = detection(im1,im2)
```

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.

```
def filtre(image,seuil):
    """prend un argument un tableau de type list à deux dimensions
       et un nombre seuil (type int) renvoie un tableau de 0 et de
       1 à 2 dimensions"""
    tab = []
    nb_lig = len(image)
    nb_col = len(image[0])
    for k in range(nb_lig):
        tab.append([])
        for j in range(nb_col):
            if image[k][j] > seuil:
                tab[k].append(1)
            else :
                tab[k].append(0)
    return tab
```

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.



Pixel central isolé

Pixel central à conserver

Test 3 est la bonne réponse.

Q10. Commenter les lignes 2 et 4, puis indiquer l'intérêt de chaque boucle `for`. Préciser l'intérêt de ce programme.


```
1    point = input("Rentrer le numéro du point litigieux à analyser")    # point est de type str
2    dirs = listdir("sequence_" + point) # concaténation entre 2 chaînes de caractères pour obtenir le nom du dossier  dirs est une liste contenant toutes les images du dossier
3    for image in dirs:          # on parcourt les images du dossier à la recherche de la dernière image repérée par son index
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 donnée par un autre traitement
10
11   for image in dirs:  #cette boucle permet de parcourir toutes les images du dossier, mais le test suivant permet de ne traiter que les images utiles à la décision par HawkEye
12       if int(image[6:11])== index_max - 3000:
13           image2=imread("sequence_"+ point +"/"+image)
14       elif int(image[6:11])> index_max - 3000:   # On s'intéresse à 2 images successives de la séquences
15           image1 = deepcopy(image2)            # on stocke l'image précédente dans image 1 (index n)
16           image2=imread("sequence_"+ point +"/"+image)# la nouvelle image (index n+1) est stockée dans image 2
17           liste_balles = traitement(image1,image2)    # on applique la fonction traitement pour repérer la balle
18       seq_balle.append(liste_balles)                  # on stocke la position de la balle
```

Intérêt de l'algorithme : récupérer la position des balles candidates du dernier échange.

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.

```
def deplacement(pos1,pos2):
    """ fonction qui prend en argument 2 listes à 2 éléments (
    type int) et qui renvoie le vecteur vitesse associé sous
        la forme d'une liste à 2 éléments (type int)"""
    return [pos2[0]-pos1[0],pos2[1]-pos1[1]]

D = deplacement(pos1,pos2)
Lx = 2*abs(D[0])+1
Ly = 2*abs(D[1])+1
```

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.

sujet pas clair [x,y,x,y...] ou [[x,y],[x,y],...]

10

```
def distance_quad(xc,yc,liste_balle_i):
    """prend en arguments les coordonnées du centre et une
       liste
    renvoie la liste des distance au carré"""
    liste_dist = []
    for k in range (len(liste_balle_i)//2):
        liste_dist+=[(xc-liste_balle_i[2*k])**2+(yc-
            liste_balle_i[2*k+1])**2]
    return liste_dist
```

16

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`.

```
def cherche_balle(xc,yc,Lx,Ly,liste_balle_i):
    """on cherche une balle dans une zone rectangulaire
       fonction qui renvoie une liste à 2 éléments """
    balle = [None,None]# initialisation (cas où aucune balle n'
       est dans la zone de recherche)
    distance_min = (Lx//2)**2 + (Ly//2)**2  # initialisation
    liste_dist = distance_quad(xc,yc,liste_balle_i)

    for k in range(len(liste_dist)):
        if ((xc-liste_balle_i[2*k])**2 <= (Lx//2)**2 ) and ((yc
            -liste_balle_i[2*k+1])**2 <= (Ly//2)**2) and (
            liste_dist[k] < distance_min):
            distance_min = liste_dist[k]
            balle = [liste_balle_i[2*k], liste_balle_i[2*k+1]]
    return balle
```

16

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.

```
def traj_balle(vit_init,seq_balle):
    """renvoie la trajectoire de la balle (liste de listes de 3
       éléments)"""
    #initialisation
    pos = seq_balle[0]      # position initiale
    traj = [pos]            # initialisation de traj (liste qui sera
       renvoyée)
    dep = vit_init          #initialisation (s'il n'y pas de balle
       dans la 1ere zone de recherche)
    Lx = 2*abs(vit_init[0]) + 1
    if Lx <3:   # Test : on vérifie que Lx >3;
        Lx=3
    Ly = 2*abs(vit_init[1]) + 1
    if Ly <3:
        Ly=3
    xc = pos[0] + vit_init[0] # centre du rectangle
    yc = pos[1] + vit_init[1]

    for k in range(1,len(seq_balle)): # on balaie les diffé
       rentes images en commençant par l'index 1
        balle = cherche_balle(xc,yc,Lx,Ly,seq_balle[k])
        traj += [balle]         # concaténation avec la position
           de l'image courante (index k)
        pos_prec = pos  #on met à jour les différentes listes
        pos = balle
        dep = deplacement(pos_prec,pos)
        Lx = 2*abs(dep[0]) + 1
        if Lx <3:
            Lx=3
        Ly = 2*abs(dep[1]) + 1
        if Ly <3:
            Ly=3
        xc = pos[0] + dep[0]
        yc = pos[1] + dep[1]

    return traj
```

2 pas de balle ??

4

Q12. Indiquer les limites ou défauts de l'algorithme étudié dans cette sous-partie.
Problème quand aucune image n'est trouvée dans la zone de recherche sur plus de 2 images successives ; choix arbitraire quand plusieurs balles au minimum ; forme de la zone de recherche ; limite de la modélisation ; ...

6

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]`.

```
def pos_loc(e,f,x1,x2,y1,y2):
    z1i=e/(x1-x2)*f
    y1i=e/(x1-x2)*y1
    x1i=e/(x1-x2)*x1
    pos1i=[x1i,y1i,z1i]
    return pos1i
```

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.

```
def pos_glo(posli,T1,Rx1,Ry1):
    posrotx1=dot(Rx1,posi1)
    posroty1=dot(Ry1,posrotx1)
    postran1=posroty1 + T1
    return postran1
```

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**.

```
def traj3D(coord_loc,T1,Rx1,Ry1):
    n=shape(coord_loc)[0]
    m=shape(coord_loc)[1]
    coord_glo=zeros((n,m))
    for i in range(n):
        coord_glo[i,:]=pos_glo(coord_loc[i,:],T1,Rx1,Ry1)
    return coord_glo
```

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 2 éléments `impact=[x_imp,z_imp]`.

```
def det_impact(coord_glo,eps):
    n=shape(coord_glo)[0]
    for i in range(1,n-1):
        if (coord_glo[i,1]>=0.033 and coord_glo[i,1]<0.033+eps):
            return [coord_glo[i,0],coord_glo[i,2]]
        elif ((coord_glo[i+1,1]-coord_glo[i,1]>0) and (coord_glo[i
            -1,1]-coord_glo[i,1]>0)):
            return [(coord_glo[i,0]+coord_glo[i+1,0])/2,(coord_glo[
                i,2]+coord_glo[i+1,2])/2]
        else :
            impact=[None, None]
    return impact
```

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.

```
def res_final(impact,l,L):
    x=impact[0]
    z=impact[1]
    if ((x<0) or (x>L) or (z<0) or(z>l)):
        res='OUT'
    else :
        res='IN'
    return res
```


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.

```
def vis_traj3D(coord_glo):
    x=coord_glo[:,0]
    y=coord_glo[:,1]
    z=coord_glo[:,2]
    gca(projection='3d').plot(x, y, z)
    show()
```

Q19. Rappeler la définition et l'intérêt d'une clé primaire.
Une clé primaire est une contrainte d'unicité qui permet d'identifier de manière unique un enregistrement dans une table.

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

```
SELECT id FROM MATCHS WHERE joueur1 = "Federer" OR joueur2 = "Federer"
```

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

```
SELECT MAX(nombre) FROM POINTS WHERE 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".

```
SELECT fichier FROM POINTS JOIN MATCHS ON POINTS.mid=MATCHS.id WHERE
nom = "Federer-Murray"
```

ou

```
SELECT fichier FROM POINTS WHERE mid = (SELECT id FROM MATCHS WHERE nom
= "Federer-Murray"))
```

Questions		Points	Remarques
Question 1		4	du/dt et dv/dt (2pt), dx/dt etdy/dt (2pt)
Question 2		6	CI (2pt), pas T/N (2pt), Euler (2pt)
Question 3		6	2 points par réponse
Question 4		4	calcul (2pt), AN (2pt)
Question 5		4	calcul (2pt), AN (2pt)
Question 6		4	déf (2pt), inconvénients (2pt)
Question 7		12	recup dim (2pt), init (2pt), boucle (4pt), calcul (2pt) et return (2pt)
Question 8		12	recup dim (2pt), init (2pt), boucle (4pt), calcul (2pt) et return (2pt)
Question 9		6	2 pt /schéma, test (2pt)
Question 10		8	commentaire lignes 2 et 3 (2pt), chaque boucle for 2pt, intérêt 2pt
Question 11	a	8	type (2pt), déplacement (4pt), instruction (2pt)
	b	10	boucle for (4pts), distance au carré (4pt), return (2pt)
	c	16	initialisation (2pt), distance_quad (2pt), boucle for (4pt), test 1 (2pt),test meilleure balle (2pt), return (4pt)
	d	16	initialisation Lx Ly (2pt), boucle for (2pt), fonction cherche_balle (2pt), mise à jour de xc, yc, et vitesse (4pt), test Lx Ly (2pt), pas de balle (2pt) return (2pt)
Question 12		4	2pt par défaut/limite
Question 13		6	calcul des coorodnnées (2pt), consctruction de la liste (2pt), return (2pt)
Question 14		6	
Question 15		8	dimension 2pt, boucle for 4pt, retour 2pt
Question 16		12	dimension 2pt, boucle 4pt, test 4pt, retour 2pt
Question 17		8	dimension 2pt, test 4pt, retour 2pt
Question 18		6	
Question 19		4	déf + intérêt (2+2 pt)
Question 20		6	Select, From Where (2pt), OR (2pt), Requête correcte (2pt)
Question 21		6	Select, From Where (2pt), Max (2pt), Requête correcte (2pt)
Question 22		8	Select, From Where (2pt), jointure (4pt), Requête correcte (2pt)
SOIN et rédaction		10	Présentation
		200	