Dans ce TP, nous proposons dans un premier temps d'étudier le fonctionnement (influence des paramètres, limitations, variantes) de différentes méthodes de détection ou d'estimation de mouvement ainsi qu'un suivi par filtre de Kalman. Le code source de ces méthodes vous est donné et des questions / tests vous sont proposées pour vous guider dans cette étude mais d'autres tests peuvent être effectués en plus ou d'autres séquences utilisées.
Dans un second temps, il vous est demandé d'implémenter le filtre particulaire en Matlab en suivant scrupuleusement l'énoncé. Vous devez en particulier utiliser les mêmes noms de variables, matrices, fonctions, *etc.* quand ceux-ci sont précisés dans l'énoncé afin que le code qui vous est fourni puisse fonctionner avec vos ajouts.
N'hésitez pas à utiliser la documentation de Matlab en tapant `help fonction` où `fonction` est la fonction sur laquelle vous voulez des informations. Vous pouvez aussi vous aider des transparents de cours ou des articles scientifiques décrivant les différentes méthodes pour avoir plus d'informations sur les paramètres ou les méthodes.

Travail de rendu demandé : à l'issu des 2 séances de TP, vous devrez envoyer une archive `zip` qui contiendra :
- Un rapport **au format pdf uniquement** contenant votre analyse de l'approche, obtenue par les tests que vous aurez effectués. Le nombre de pages n'est pas un critère de jugement, tant que le rapport contient une analyse aboutie montrant que vous avez compris l'approche. Plus l'analyse sera aboutie, meilleure sera la note !
- L'ensemble de vos sources (commentées !) avec un petit script permettant de lancer facilement votre programme.

Tout plagia de code source ou de rapport sera sévèrement sanctionné.

Les données ainsi que les scripts Matlab peuvent être récupérés dans le dossier `commun/dietenbeck/5AI03/TP`.

EXERCICE 1: DÉTECTION DE MOUVEMENT

L'objectif de cet exercice est d'étudier le fonctionnement de méthodes de détection de mouvement soit par différences d'images soit reposant sur des modèles statistiques ([ElGammal 2002] et [Olivier 2000]).
Le script `ex1.m` permet de choisir la séquence de travail (variable `seqNum`, ligne 8). Il définit ensuite une variable `typeDetect` permettant de choisir entre les différentes méthodes de détection ainsi que les paramètres nécessaires au fonctionnement de chaque méthode (si elle en a).

1. **Différences d'images :** on s'intéresse dans un premier temps à la détection de mouvement par différences d'images (fond connu, image précédente et filtrage temporel) et on s'assurera que `typeDetect = 1` (ligne 28).
   (a) Lancer le script sur différentes séquences et observer le résultat de la détection par fond connu (`typeDiff = 1`, ligne 33) et différence d'images successives (`typeDiff = 2`). Comment se comportent ces méthodes en cas de mouvement dans le fond ou d'objet statique ? Les objets sont-ils bien détectés ? Observe t'on un phénomène de rémanence, d'ombre portée ?
   (b) Comment évolue la détection par différence d'images successives en fonction du pas de temps `tStep` (ligne 33) ?
   (c) Refaire ces tests avec une détection par fond connu et mis à jour (`typeDiff = 3`).
   (d) Comment évolue la détection en fonction de la variable d'apprentissage α (ligne 34) ?

2. **Modèles statistiques :**
   (a) *Estimateur de densité par noyaux*, [ElGammal 2002]
   - Mettre la variable `typeDetect` à 2 et relancer le script sur différentes séquences. Mêmes questions qu'en (1.a).
   - Comment évolue la détection en fonction du nombre de frames `nEG` (ligne 37) utilisées pour l'estimation de la densité ?

   Remarque : pour la détection des `nEG` premières frames, l'estimation de la densité de probabilité se fait sur les frames 1 à `nEG`.

   (b) *Apprentissage (ACP)*, [Olivier 2000]
   - Mettre la variable `typeDetect` à 3 et relancer le script sur différentes séquences. Mêmes questions qu'en (1.a).
   - Comment évolue la détection en fonction
     - du nombre de frames `nACP` (ligne 40) utilisées pour l'apprentissage ?
     - du pourcentage `M` (ligne 41) d'eigenbackgrounds conservés ?

     Vous pourrez vous aider de l'image moyenne ainsi que des eigenbackgrounds pour expliquer ces phénomènes.

   Remarque : l'apprentissage (ACP) se fait sur les `nACP` premières frames et n'est pas mis à jour en fonction des frames suivantes.

EXERCICE 2: ESTIMATION DE MOUVEMENT

L'objectif de cet exercice est d'étudier différentes méthodes d'estimation de mouvement (block-matching, [Lucas & Kanade 1981], [Horn & Schunck 1981]) et leurs variantes (4 step search, [Brühn 2005]). On s'intéressera d'abord à des images simples (binaires ou bruitées) pour étudier certaines limitations des algorithmes puis à des images réelles pour étudier l'influence des paramètres et les temps de calcul des méthodes.
Le script `ex2.m` permet de choisir entre des images images binaires, bruitées ou réelles (variables `isReal`, `isNoise` et `isIntensity`). Il définit ensuite une variable `typeEstim` permettant de choisir entre les différentes méthodes d'estimation ainsi que les paramètres nécessaires au fonctionnement de chaque méthode.

1. **Block matching, Exhaustive search :** pour cette partie, on s'assurera que `typeEstim = 1` (ligne 24) et `N = 1` (ligne 29).
   (a) *Image binaire :* on s'assurera que `isReal = 0` (ligne 8), `isNoise = 0` (ligne 17) et `isIntensity = 0` (ligne 18). La fonction `getImgsBM` génère alors 2 images binaires dans lesquelles se déplace un carré blanc. Le déplacement de ce carré est définit par la variable `vBlock` en ligne 16.
   — On appelle bloc une matrice 2D de taille (2dimB+1, 2dimB+1) et ROI la zone de recherche de taille (2dimR+1, 2dimR+1) centrée autour de ce bloc. `dimB` et `dimR` sont des variables définies en ligne 27 et 28.
   Lancer le script pour estimer le mouvement. Est il correct ? Si non, en quel(s) point(s) y a t'il des erreurs ? Expliquer.
   — Pourquoi le mouvement dans les zones homogènes est-il non nul ? Modifier la fonction `blockMatching` pour corriger ce problème.
   — Faire varier la taille du bloc `dimB` et étudier l'influence de ce paramètre sur l'estimation (qualité, temps de calcul, ...).
   — Faire varier le mouvement (variable `vBlock`) et étudier l'influence de la taille de la ROI `dimR`.

   Remarque : Pour trouver les raisons pour lesquelles les estimations sont erronées, n'hésitez pas à visualiser le bloc recherché et la zone de recherche et à utiliser des points d'arrêt (breakpoint) dans voir code Matlab.

   (b) *Images texturées*
   — Mettre la variable `isNoise` à 1 pour obtenir des images texturées.
   — Estimer et afficher le mouvement. Est il correct ? Si non, en quel(s) point(s) y a t'il des erreurs ? Expliquer.
   — Faire varier la taille du bloc ou de la ROI ainsi que le déplacement et étudier l'influence de ces paramètres sur l'estimation (qualité, temps de calcul, ...).
   — Mettre la variable `isIntensity` à 1 pour simuler une variation d'illumination.
   — Estimer et afficher le mouvement. Est il correct ? Si non, en quel(s) point(s) y a t'il des erreurs ? Expliquer.
   — Comment faire pour conserver une estimation correcte malgré le changement d'illumination ?

2. [Lucas & Kanade 1981]
   (a) *Images binaires :* on remettra `isNoise = 0` (ligne 17) et `isIntensity = 0` (ligne 18) afin de travailler sur des images binaires et on choisira `typeEstim = 3` pour l'estimation par la méthode de [Lucas & Kanade 1981].
   — Calculer et afficher le mouvement estimé. Le script s'exécute t'il sans erreur ? Si non, quelles en sont les causes ?
   — Le mouvement estimé est-il correct ? Si non, en quel(s) point(s) y a t'il des erreurs ? Expliquer.
   — Faire varier la taille `sW` (ligne 35) de la fenêtre et observer l'influence du paramètre.

   (b) *Images texturées*
   — Mettre la variable `isNoise` à 1 pour obtenir des images texturées. Estimer et afficher le mouvement. La fonction s'est elle exécutée sans erreur ?
   — Est il correct ? Si non, en quel(s) point(s) y a t'il des erreurs ? Expliquer.
   — Remplacer la fenêtre de pondération uniforme par une pondération gaussienne (`typeW = 1` en ligne 36). Quelle différence peut on observer avec le mouvement estimé sans pondération (*i.e.* avec `typeW = 0`) ?

3. **Images réelles et comparaison :** on va désormais s'intéresser à des images réelles et on choisira `isReal = 1` (ligne 8)
   (a) *Appariement de blocs*
   — Comparer les méthodes de block matching (`typeEstim = 1`) et de 4 Step Search (`typeEstim = 2`). Le mouvement obtenu est-il le même ? Quand est-il du temps de calcul ?

   (b) *Flux optique*
   — Comparer les méthodes de [Lucas & Kanade 1981] (`typeEstim = 3`), de [Horn & Schunck 1981] (`typeEstim = 4`) et de [Brühn 2005] (`typeEstim = 5`). Le mouvement obtenu est-il le même ? Quand est-il du temps de calcul ? Et comparé aux méthodes d'appariement de blocs ?
   — Étudier l'influence du nombre d'itérations et de α sur la qualité de l'estimation par [Horn & Schunck 1981] et [Brühn 2005].

EXERCICE 3: FILTRE DE KALMAN : SUIVI DU MYOCARDE

Le but de cet exercice est de suivre le ventricule gauche (ou myocarde) dans des séquences échographiques acquises en petit axe. On s'intéressera principalement à l'endocarde (interface entre le muscle et le sang). Afin de suivre le myocarde, on dispose

— d'une estimation du mouvement en tout point de la séquence (variables u et v)
— de la référence d'un cardiologue dont on se servira pour initialiser le modèle de Kalman ainsi que pour valider le suivi. Il s'agit des variables refEndo (pour l'endocarde) et refEpi (pour l'épicarde).

On suppose que l'endocarde (ou l'épicarde) peut être modélisé par une ellipse de paramètres $\boldsymbol{\lambda} = \{c_x, c_y, r_x, r_y, \theta\}$.

Le script `ex3.m` permet de charger les données et de suivre l'endocarde (ou l'épicarde) au cours du temps en appliquant un simple tracking, c'est à dire en ajoutant à chaque point du contour à t le mouvement estimé entre t et $t + \Delta t$ pour obtenir la nouvelle position du point à $t + \Delta t$ ainsi que par filtre de Kalman. Enfin, le script affiche le contour de référence et les contours obtenus à l'instant t et calcule plusieurs distances entre eux.

1. Lancer le script et conclure sur les performances du tracking (notamment par rapport à l'hypothèse de forme elliptique du contour).
2. **Filtre de Kalman :**
   (a) Le filtre de Kalman sera supposé à vitesse constante. L'état sera donc $\mathbf{x}_t = \{c_x, c_y, r_x, r_y, \theta, Vc_x, Vc_y, Vr_x, Vr_y, V_\theta\}$ où Vc_x correspond à la vitesse d'évolution de c_x. L'observation sera $\mathbf{y}_t = \{c_x, c_y, r_x, r_y, \theta\}$.
   Donner les équations d'état et d'observation ainsi que les différentes matrices du modèle de Kalman.
   (b) Comparer les résultats (distances entre les contours) obtenus par tracking et filtre de Kalman pour les 2 séquences disponibles et pour les interfaces (endocarde et épicarde). Conclure sur l'intérêt du filtre de Kalman.
   (c) Faire varier les variances des bruits (variables `s2v` (bruit d'état) et `s2n` (bruit de mesure), ligne 16) et étudier leurs influences sur les résultats de segmentation. Étudier en particulier l'influence du rapport `s2v/s2n`.
   (d) Étudier l'influence du nombre de points `nbPts` (ligne 15) représentant le contour sur les résultats de segmentation.
3. **Suivi du myocarde :** on souhaite maintenant suivre les 2 interfaces du ventricule gauche. Quels seraient les avantages et inconvénients des solutions suivantes (**aucun** code n'est demandé) :
   — 2 filtres de Kalman (un pour l'endocarde, un pour l'épicarde)
   — 1 filtre de Kalman regroupant les paramètres des 2 ellipses
   — (optionnel) 1 filtre de Kalman contenant les paramètres d'une ellipse ainsi que l'épaisseur myocardique

EXERCICE 4: FILTRAGE PARTICULAIRE

Dans cet exercice, nous cherchons à estimer la position (x, y) d'un objet à chaque instant dans une séquence d'images. Pour cela, nous allons le modéliser par l'histogramme couleur de la région autour de la position de l'objet. Pour estimer cette position, un ensemble de N particules va être propagé à chaque instant, chaque particule étant une position possible de l'objet dans l'image courante. Les particules seront pondérées selon leur vraisemblance par rapport au modèle de l'objet. Enfin, la position de l'objet sera estimée par la somme pondérée des particules. Un rééchantillonnage des particules sera ensuite effectué.

1. Le script `ex4.m` permet de charger les données et d'initialiser le filtrage en traçant un rectangle autour de l'objet d'intérêt. Il génère aussi N particules au centre du rectangle et leur affecte un poids de $1/N$ (variable `omegaT`). L'ensemble des paramètres de réglage de la méthode (N, σ, *etc.*.) sont définis au début du programme (lignes 12 à 16). Vous pourrez les faire varier pour étudier leur influence sur le suivi.

   Nous allons maintenant écrire l'algorithme général, traitant à chaque instant une nouvelle image, propageant, corrigeant, normalisant les particules, puis estimant la nouvelle position de l'objet et rééchantillonnant l'ensemble de particules. Il est conseillé de valider le code dans un premier temps entre l'image 1 (initialisation) et l'image 2. Faites donc en sorte que la boucle ne parcourt pas l'ensemble des images de la séquence.

2. **Propagation des particules :** Les particules $\{\mathbf{x}_t^i\} = \{x_t^i, y_t^i\}_{i=1}^N$ à l'instant t, stockées dans la variable `xT`, sont propagées selon la fonction de transition $p(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i)$. On utilisera une distribution gaussienne pour générer les nouvelles positions (on parle de marche aléatoire gaussienne).
   (a) Écrire la commande qui, à partir des positions des particules stockées dans `xT`, génère de nouvelles positions pour ces particules, que l'on stockera directement dans `xT`.
   (b) Afficher les particules que vous avez générées à la question précédente. En particulier, vous testerez plusieurs valeurs de σ pour la gaussienne.

   Fonction Matlab utile :
   — `randn` pour générer aléatoirement des valeurs selon $\mathcal{N}(0, 1)$. On rappelle que si $x \sim \mathcal{N}(0, 1)$ alors $y = \mu + \sigma x \sim \mathcal{N}(\mu, \sigma)$.
   — l'affichage des particules pourra être fait en utilisant le code suivant :

```
imagesc( img ); axis image; axis off;
hold on; plot( xT(1, :), xT(2, :), 'r+' ); hold off;
```

3. **Correction et normalisation :** L'étape de correction vise à affecter à chaque particule un poids. Pour cela, on va comparer la région englobant la particule (définie par sa position), à celle du modèle que l'on a de l'objet (venant de sa localisation manuelle dans la première image de la séquence). On a besoin pour cela de calculer les histogrammes normalisés de chacune des particules $\mathbf{h}_t^i$, $i = 1, \ldots, N$. Le calcul du poids est donné par :

$$\omega_t^i = \omega_{t-1}^i e^{-\lambda d^2(\mathbf{h}_{ref}, \mathbf{h}_t^i)}$$

   où d^2 est la distance de Bhattacharrya entre les histogrammes et λ un paramètre à fixer empiriquement définissant l'étalement de la vraisemblance.
   (a) Écrire la fonction `corrigerParticules( img, h_ref, xT, omegaT, lambda, l, h, N, nbins )` qui étant donnés l'image courante `img`, l'histogramme normalisé modèle `histo_ref`, les positions des particules stockées dans `xT`, le tableau de poids `omegaT`, la largeur `l` et la hauteur `h` de la région, le nombre de particules `N`, le nombre de bins de l'histogramme `nbins`, renvoie les nouveaux poids des particules.
   Remarque : On pourra utiliser les fonctions `getHisto` et `dBhattacharrya` pour le calcul de l'histogramme et de la distance de Bhattacharrya respectivement.
   (b) Ecrire une ligne de commande qui normalise les poids obtenus au retour de la fonction `corrigerParticules`.

4. **Estimation :** L'estimation est la somme pondérée des particules (leurs poids ayant été normalisés).
   (a) Écrire une ligne de commande qui calcule `lig_est` et `col_est`, respectivement le numéro de ligne et de colonne du centre de la boîte estimée comme englobant l'objet à suivre.
   (b) Décommenter les lignes permettant d'afficher la boîte sur l'image. Tester le programme total sur les deux premières images. Faire varier les valeurs des paramètres `nbBin`, `lambda`, `nbPart`, `sigma` pour commencer à comprendre le rôle de chacun.

5. **Rééchantillonnage :** L'étape de rééchantillonnage vise à supprimer les particules de faible poids (qui n'aident pas pour le suivi, puisque peu vraisemblables) et dupliquer celles de fort poids. Pour cela, même s'il en existe d'autres, nous allons utiliser la méthode dite de rééchantillonnage multinomial (vue en cours). Nous rappelons ci-dessous l'algorithme général :
   — Calculer le tableau de somme cumulée des poids (normalisés) des particules.
   — Répéter N fois :
      — Tirer aléatoirement $k \sim \mathcal{U}(0, 1)$ (selon une loi uniforme entre 0 et 1).
      — Trouver le premier indice i dans la tableau de la somme cumulée des poids tel que $C(i-1) < k \leq C(i)$.

— Sélectionner la particule d'indice i en la rangeant dans un tableau.

(a) Écrire la fonction `selectionParticules( xT, omegaT, N )` qui étant donnés les tableaux des positons et des poids des particules (respectivement `xT` et `omegaT`) et le nombre particules `N`, renvoie le tableau `Pselect` contenant les positions des particules après rééchantillonnage (*i.e.* celles qui ont été dupliquées).

(b) Afficher les particules avant et après rééchantillonnage, en utlisant une couleur différente. Que constatez-vous ?
Fonctions Matlab utiles :
— `cumsum` calcule une somme cumulée.
— `rand` génére aléatoirement selon une loi uniforme.
— `find` trouve les indices d'une valeur.

6. Lancer le suivi sur toute la séquence et évaluer qualitativement le résultat obtenu. Essayer votre algorithme sur plusieurs séquences et avec différentes valeurs de paramètres.