TP Vision par Ordinateur : Tracking

Le but de ce TP est de se familiariser avec le suivi d'objets. Le sujet est découpé en 5 étapes, et est très guidé : la structure du programme est déjà définie, et les noms de variables et de fonctions sont déjà choisis, vous devrez les respecter. A chaque étape vous rajouterez les éléments mentionnés, et, dans le cas des fonctions, les compléterez selon les commentaires. Ces éléments sont ceux dans les cadres.

A - Lire une vidéo

La première étape est bien entendu de lire une vidéo.
La lecture de flux vidéos, que ce soit depuis un fichier ou depuis la webcam, se fait via la classe cv::VideoCapture

http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html#videocapture

Chaque image que vous allez extraire du flux vidéo sera traitée par une suite de fonctions.
Pour éviter de la passer en paramètre à chaque fois, on va la stocker en variable globale :

```
cv::Mat nextInput;
```

Nous aurons également besoin d'une fonction de dessin, qui affichera l'image, ainsi que les différentes choses à dessiner dessus (zone d'intérêt, points trackés, etc…).

```
void draw(){
        cv::Mat img = nextInput.clone();
        // dans les prochaines étapes, c'est ici que l'on mettra les choses a dessiner
        cv::imshow("input", img);
        //si il y a un call back de souris a mettre, ca sera ici
}
```

La fonction principale, qui effectuera le traitement sur le flux vidéo, sera appelée depuis le main.

```
void video(char* videoname){
        cv::VideoCapture cap;
        //si videoname n'est pas null, ouvrir la video dans cap, sinon ouvrir la camera 0

        //si cap n'est pas ouvert, quitter la fonction

        //recuperer une image depuis cap et la stocker dans nextInput

         //tant que nextinput n'est pas vide
        // - >  faire les traitements sur l'image (prochaines étapes)
        // - > appeler la fonction de dessin
        // - > recuperer une nouvelle image et la stocker dans nextInput
        // - > attendre 10ms que l'utilisateur tape une touche, et quitter si il le fait
}
```

B - Detecter les points d'intérêt

Maintenant que nous avons un flux de données, nous pouvons nous attaquer a trouver les points interessants à tracker. Une méthode toute faite existe dans OpenCV pour cela : http://docs.opencv.org/modules/imgproc/doc/feature_detection.html#goodfeaturestotrack

Nous allons stocker les points détectés dans une variable globale.

```
std::vector<cv::Point2f> prevPoints;
```

Nous allons maintenant créer une fonction de détection de points d'intérêt :

```
void detectPoints(cv::Mat & img){
        //utiliser la fonction goodFeaturesToTrack pour detecter les points d'interet
        // dans img et les stocker dans prevPoints
}
```

Vous devez ensuite appeler cette fonction depuis votre fonction principale, et dessiner chaque point détecté par un petit cercle dans votre fonction de dessin.

C - Suivre les points d'intérêt

L'étape suivante consiste à suivre nos points d'intérêt d'une image à l'autre, c'est à dire connaître leur nouvelle position. Nous allons donc avoir besoin d'une deuxième image, et d'un nouvel ensemble de points, correspondant aux points détectés.

```
cv::Mat prevInput;
std::vector<cv::Point2f> nextPoints;
```

Nous allons maintenant créer une fonction qui suis nos points : il faut trouver les correspondances des points d'intérêt de **prevInput**, stockés dans dans **prevPoints**, dans la nouvelle image **nextInput**, et les stocker dans **nextPoints**.

Une fonction OpenCV existe pour faire le suivi de points :
http://docs.opencv.org/modules/imgproc/doc/feature_detection.html#goodfeaturestotrack

```
void trackPoints(){
        //si prevInput n'est pas vide
        // - > mettre dans prevPoints le contenu de nextPoints
        // - > si prevPoints ne contient pas assez de points,
        //      appeler la fonction de detection de points sur prevInput
        // - > si prevPoints ne contient toujours pas assez de points,
        //      quitter la fonction
        // - > calculer a l'aide de calcOpticalFlowPyrLK  les nouvelles positions
        //       (nextPoints) dans la nouvelle image (nextImage)
        // -> supprimer  les points non suivis des deux listes de points

        //cloner nextInput dans prevInput
}
```

Pour la partie "supprimer les points non suivis des deux listes de points", je vous donne cette fonction.

```
std::vector<cv::Point2f> purgePoints(std::vector<cv::Point2f>& points,
                                     std::vector<uchar>& status){
        std::vector<cv::Point2f> result;
        for(int i = 0; i < points.size(); ++i){
                if(status[i]>0)result.push_back(points[i]);
        }
        return result;
}
```

Note : la fonction ***calcOpticalFlowPyrLK*** *se trouve dans le module* ***video****, penser a ajouter l'include qui va bien et de faire les ajouts dans la configuration.*

Remplacez ensuite l'appel a detectPoints par un appel a trackPoints dans la fonction principale.

Dans la fonction de dessin, replacez le dessin des points de prevPoints par ceux de nextpoints, avec en plus une ligne entre les points correspondant entre les deux listes de points.

D - Sélectionner une zone d'intérêt (ROI)

Nous avons une première version qui marche, mais nous n'allons pas nous arrêter la : nous allons maintenant sélectionner la zone a suivre dans l'image, grâce a la souris.

Vous aurez besoin de deux variables globales :

```
cv::Rect roi;
cv::Point start(-1,-1);
```

Le callback de souris a utiliser pour sélectionner une zone a la souris peut être celui-ci :

```
void CallBackFunc(int event, int x, int y, int flags, void* userdata)
{
    if  ( event == cv::EVENT_LBUTTONDOWN )
    {
      start = cv::Point(x,y);
      roi = cv::Rect();
      prevPoints.clear();
      nextPoints.clear();
    }
    else if ( event == cv::EVENT_MOUSEMOVE )
    {
       if(start.x>=0){
                        cv::Point end(x,y);
                        roi = cv::Rect(start, end);
                  }
    }else if( event == cv::EVENT_LBUTTONUP){
                      cv::Point end(x,y);
                      roi = cv::Rect(start, end);
                      start = cv::Point(-1,-1);
   }
}
```

Modifier la fonction de détection de points d'intérêt pour qu'elle ne cherche les points que dans **roi** (utilisez un masque). Si roi a une aire insuffisante (**roi.area()<10**) ou si la zone est en train d'être sélectionnée (**start.x >= 0**), aucune détection de points n'est faite.

Créez une méthode qui met à jour la zone d'intérêt (si non vide) en prenant le rectangle englobant tous les points contenus dans nextPoints.

```
void updateROI(){
      //si roi est vide ou si une selection est en cours, on quitte direct

      //sinon, on calcule le rectangle englobant notre ensemble de points stockés
      // dans nextPoints, et on l'assigne a roi
}
```

Modifiez la fonction de dessin pour afficher également la zone d'intérêt si définie ou si en cours de sélection.

E - Filtrer les outliers

Vous pouvez terminer en filtrant les points qui se comportent bizarrement. Libre a vous d'essayer différentes approches.
Quelques fonctions qui peuvent vous inspirer :
http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html#estimaterigidtransform
http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#pointpolygontest