Documentation J4K Java Library

Robin Shin et Thibaud Lemaire

PACT 2015-2016

Installation

Tout d'abord, il est *impératif* d'avoir la version 1.8 du SDK Kinect et non la 2.0 car celle-ci est inexploitable (problèmes de pilotes).

Téléchargement du fichier .jar

Sur : http ://research.dwi.ufl.edu/ufdw/download.php, télécharger ufdw.jar.

Intégration avec Eclipse

1. Ouvrir Eclipse, aller dans Project > Properties et aller dans l'onglet Java Build Path
2. Cliquer sur le bouton "Add External JARs..." et choisir le chemin vers le fichier ufdw.jar précédemment téléchargé.
3. Déplacer le fichier "ufdw_j4k_32bit.dll" ou "ufdw_j4k_64bit.dll" en fonction de la machine utilisée à la racine du projet Java pour éviter les problèmes de dll manquantes.
4. On peut désormais importer les fichiers de la librairie gr ?ce ? la commande "import edu.ufl.digitalworlds.j4k.*".

Recommandations pour l'intégration avec git

1. Tout d'abord, il ne faut surtout pas ignorer le fichier .classpath au risque d'avoir des problèmes avec git.
2. Dans le fichier .classpath, le chemin vers ufdw.jar doit ?tre relatif et non absolu au risque de conflits dès qu'un utilisateur souhaite faire un push ou un pull.

Ajout d'un projet de démonstration sous Eclipse

1. Ouvrir Eclipse, aller dans File > Import... et sélectionner Git > Projects from Git
2. Sélectionner URI puis cliquer sur "Next"
3. Copier http ://research.dwi.ufl.edu/git/j4kdemo dans l'espace dédié ? l'URI et cliquer sur "Next" autant de fois que nécessaire, puis "Finish" : un nouveau projet "j4kdemo" est crée.

Création de l'objet Kinect

Initialisation

`public int start(int flags);`

Initialise la Kinect, les données, le squelette, etc... L'entier flags passé en paramètre permet d'ajouter des options, en fonction des besoins du projet. Par exemple, `flag = COLOR` initialise la Kinect avec une image en couleur, `DEPTH` permet d'avoir la 3D et `flag = SKELETON` permet d'exploiter le squelette. On peut enfin cumuler ces options grâce au séparateur `|`.

Méthodes

`public void onSkeletonFrameEvent(boolean[] skeleton_tracked, float[] positions, float[] orientations, byte[] joint_status);`

Méthode appelée lorsqu'un nouveau squelette est reçu. Cette méthode remplace le squelette de l'attribut associé de type Skeleton, crée un événement et l'envoie à tous les modules via un système de Listeners.

Il est important d'exploiter le tableau de booléens `skeleton_tracked` pour savoir quel squelette a été reçu sous peine d'un comportement de la Kinect assez imprévisible.

`public void onDepthFrameEvent(short[] arg0, byte[] arg1, float[] arg2, float[] arg3);`

Cette méthode est appelé lorsque le depthFrame est recu.

`public void stop();`

Cette méthode permet d'arrêter la Kinect et de fermer le flux précédemment ouvert.

Comment les événements sont-ils récupérés ?

Les événements sont récupérés via un système de Listeners.

Qu'est-ce qu'un listener ?

Un listener est une instance d'une classe qui possèdes certaines méthodes qui sont destinées à être appelées par un gestionnaire d'événement. Une classe de listener doit hériter de la classe 'EventListener'. Nous allons nous servir de ces listeners pour faire les transitions entre tous les modules de notre projet.

Utilisation dans notre projet

La classe KinectEvent

Nous avons besoin de créer une nouvelle classe KinectEvent. En effet, ce sera un objet de la classe KinectEvent qui sera récupéré par les autres modules. Cette classe possède un attribut Skeleton et un attribut long représentant le temps. Elle possède également deux getters permettant de récupérer ces attributs, et permettant ainsi à la classe ayant reçu un événement d'exploiter le squelette.

L'interface KinectListenerInterface

Cette interface va être implémentée par toutes les classes ayant besoin d'écouter la classe Kinect. Cette interface ne contient qu'une méthode de signature `void skeletonReceived(KinectEventInterface e);` qui va devoir être implémentée à chaque fois qu'une classe implémente KinectListenerInterface. C'est cette méthode qui sera appelée dès qu'un KinectEventInterface sera reçu.

Comment le squelette est-il manipulé ?

On suppose avoir un objet de type Skeleton appelé squelette. Alors la classe Skeleton implémentée par la J4KSDK fournit directement des méthodes de signatures `public float get3DJointX(int joint_id);`, `public float get3DJointY(int joint_id);`, `public float get3DJointZ(int joint_id);` permettant de récupérer respectivement les coordonnées (x, y, z) du squelette. Enfin, la classe Skeleton contient plusieurs constantes de classes permettant de tracker une partie du corps bien précise en remplaçant l'entier `joint_id` par une des constantes de classes suivantes :

Identifiant	Entier associé
SPINE_BASE	0
SPINE_MID	1
NECK	2
HEAD	3
SHOULDER_LEFT	4
ELBOW_LEFT	5
WRIST_LEFT	6
HAND_LEFT	7
SHOULDER_RIGHT	8
ELBOW_RIGHT	9
WRIST_RIGHT	10
HAND_RIGHT	11
HIP_LEFT	12
KNEE_LEFT	13
ANKLE_LEFT	14
FOOT_LEFT	15
HIP_RIGHT	16
KNEE_RIGHT	17
ANKLE_RIGHT	18
FOOT_RIGHT	19
SPINE_SHOULDER	20
HAND_TIP_LEFT	21
THUMB_LEFT	22
HAND_TIP_RIGHT	23
THUMB_RIGHT	24
JOINT_COUNT	25

Exemple pour tracker la main droite

Il suffit de tapper les lignes de commandes :

```
int x, y, z;
x = get3DJointX(Skeleton.HAND_RIGHT);
y = get3DJointY(Skeleton.HAND_RIGHT);
z = get3DJointZ(Skeleton.HAND_RIGHT);
```

Alors les coordonnées de la main droite sont (x, y, z).