|  |
| --- |
| PST : Introduction à Google Tango |
| Résultat de recherche d'images pour "tango google" |
| Acquisition d’une image de profondeur avec Google Tango |
| https://slack-imgs.com/?c=1&url=http%3A%2F%2Fcdn2.knowyourmobile.com%2Fsites%2Fknowyourmobilecom%2Ffiles%2Fstyles%2Fgallery_wide%2Fpublic%2FArray%2Fandroid-logo.jpg%3Fitok%3DUE5uj0i7 |
| DELANGLE Flavien et SEURRE Maxime |
|  |

PST : Introduction à Google Tango

Acquisition d’une image de profondeur avec Google Tango

# Introduction :

Tango est une plateforme hardware et software développée par Google destinée à équiper les appareils mobiles (smartphones, tablettes). L’objectif est de proposer aux utilisateurs de nouvelles applications de réalité augmentée basées sur le motion tracking, la reconnaissance d’environnement et la perception de profondeur. Parmi ces applications, on retrouve par exemple la navigation indoor ou l’intégration d’éléments virtuels dans l’espace physique.

En ce qui concerne la perception de la profondeur, Tango repose sur la 2.5D (image 2D complétée par une image de profondeur) obtenue grâce à son lot de capteurs, et notamment un capteur RGB-IR (infrarouge). Des technologies comme Time of Flight (temps de vol) ou Structured Light (lumière structurée) permettent l’acquisition des données de profondeurs, tandis que le capteur RGB capte les images de couleur.

Quelques smartphones sont aujourd’hui équipés de Tango, dont le Lenovo Phab 2 Pro sur lequel nous avons travaillé.

L’objectif de notre PST était de nous familiariser avec l’environnement Tango et de proposer une application permettant d’acquérir la Depth Map (carte de profondeur), en intégrant en parallèle une librairie dédiée à la vision par ordinateur : OpenCV. A postériori, le but est de pouvoir utiliser à la fois les informations de couleur et de profondeur afin d’appliquer certains traitements d’image tels que la détection de visage, la détection de landmarks ou encore le recalage, qui sont utiles dans des problématiques comme la reconnaissance d’émotions.

Nous présenterons dans ce rapport l’application développée et ses fonctionnalités, une vue d’ensemble du fonctionnement du programme ainsi qu’un certains nombres de points sensibles que nous avons jugés utiles de mentionner, notamment afin d’assurer une meilleure transition et compréhension de notre travail.

# Présentation de l’application :

// A rajouter : ~~Présentation Tango~~ : ~~lenovo phab 2 pro~~, ~~dev sous Android Studio~~, Architecture globale (hierarchie), configuration (.mk et tout le tintouin) , annexes, La suite (mdr, parce que c’est deprecated)

# Fonctionnalités :

* Affichage en temps réel au choix de la vidéo RGB ou de la Depth Map
* Enregistrement de la vidéo RGB et de la Depth Map sur la carte SD du téléphone sous forme de fichiers PNG
* Détecteur de visage mais on est pas sur parce qu’on a potentiellement la flemme voilà

# Développement sous Android Studio :

Tango propose trois API de développement : une API Java, une API C++ et une API Unity. Nous avons dans un premier temps utilisé l’API Java qui nous paraissait la plus évidente dans un contexte de développement pour application mobile. Cependant, après un certain temps passé à découvrir l’environnement Tango, il s’est avéré qu’il nous était impossible d’accéder à la fois à la caméra de couleur et au capteur de profondeur en utilisant l’API Camera d’Android. De ce fait, la majeure partie de la logique logicielle de notre application a finalement été développée en C++. Nous avons basé notre projet sur un des exemples en C++ fournis : *cpp\_rgb\_depth\_sync\_example*.

TangoDepthMapRenderer

TangoDepthMapHelper

TangoDepthMapActivity

Storage

TangoJNINative

Java

CPP

jni\_interface

tango\_depth\_map\_app

scene

depth\_image

color\_image

camera\_texture\_drawable

*Configuration sous Android Studio :*

**Manifest**

Ne pas oublier de spécifier les permissions pour les accès à la caméra et à la carte sd/mémoire interne

**Build Files**

Android.mk :

Les fichiers C++ du projet doivent figurer dans LOCAL\_SRC\_FILES

Dans TangoJNINative – System.loadlibrary(“name”); name doit correspondre à LOCAL\_MODULE := lib + name

CVROOT : spécifier le chemin jusqu’à sdk/native/jni selon votre version d’OpenCV (PROJECT\_ROOT est défini dans userPaths.mk).

**Build.gradle(Module :app)**

Spécifiez le chemin vers votre Android.mk comme suit

externalNativeBuild {

ndkBuild {

path 'path/to/my/Android.mk'

}

}

Spécifiez le filtre sur les architectures du CPU

defaultConfig {

...

ndk {

abiFilters 'cpu\_archi\_1', 'cpu\_archi\_2'

}

}

Celles-ci doivent correspondre au Application.mk : APP\_ABI

Local.properties : spécifiez le chemin vers votre ndk et sdk dans ndk.dir et sdk.dir

**JNI**

En principe, on crée un dossier cpp dans src/main qui contiendra les fichiers C++ et leurs entêtes ainsi que Android.mk et Application.mk. Ce n’est évidemment pas une obligation, tant que la configuration est consistante dans votre projet.

# Communication Java - C++ : La JNI

## Exemple d’appel d’une fonction C++ depuis un fichier Java :

Java :

JniInterface.myJavaMethod(5.0f);

Java – JniInterface.java :

public static native void myJavaMethod(float val);

Cpp – JniInterface.cpp :

static mynamespace::MyCPPClass cppClass;

JNIEXPORT void JNICALL

Java\_your\_package\_class\_ yourJavaMethod (

JNIEnv \*, jobject , jfloat val) {

return cppClass.myCppFunction(val);

}

Cpp – MyCPPClass.cpp:

void MyCPPClass::myCppFunction(float value){

//Your function processing the value

}

## Exemple d’appel d’une fonction Java depuis un fichier C++ :

Cpp :

**// Access the class *your.package.myJavaClass***

jclass myClass = env->FindClass("your/package/myJavaClass");

if (myClass == nullptr) {

LOGE("*Class not found*");

}

else {

// **Access the default constructor with <init>**

// **()V : no arguments, void return**

jmethodID constructor = env->GetMethodID(myClass, "<init>", "()V");

if (constructor == nullptr) {

LOGE("*Constructor not found*");

}

else {

**// Create an instance of the class**

jobject myClassObject = env->NewObject(myClass, constructor);

**// Access the method yourMethod**

**// Here, (I)I means 1 parameter of type Integer, return Integer)**

jmethodID method = env->GetMethodID(myClass, "*yourMethod*", "(I)I");

if (method == nullptr) {

LOGE("*Method not found*");

}

else {

**// Call the method with the needed arguments**

env->CallIntMethod(myClassObject, method, 5);

}

}

}

Java, myJavaClass.java

int yourMethod (int someNumber) {

//Your function returning an int

}

Remarque : Il est nécessaire d’avoir accès au pointeur d’environnement pour ces fonctions (*FindClass, NewObject, etc*). Pour ce faire il est conseillé, par exemple dans une méthode *onCreate* de stocker un pointeur *JavaVM* de la manière suivante :

JavaVM\* javaVM;

env->GetJavaVM(&javaVM);

On récupèrera ensuite, au moment souhaité, le pointeur d’environnement comme suit :

JNIEnv \*env;

javaVM->AttachCurrentThread(&env, NULL);

# A l’attention des prochains groupes :

Compétences requises : Programmation orientée objet (Java et C++)

Compétences utiles : Android, OpenCV, OpenGL, du bon sens

## Informations :

Afin de build l’application à partir de votre ordinateur, il est nécessaire de modifier le fichier *PSTTango/TangoDepthMap/userPaths.mk* pour chaque utilisateur afin de spécifier le chemin absolu pointant vers votre dossier *PSTTango*.

Tous les modules locaux (ex : OpenCV) ont été placé dans le dossier *PSTTango/Modules/.* Il pourrait être intéressant de les retirer du projet Github afin de l’alléger.

## Conseils :

1) Essayez de comprendre les principes de base de la JNI (Java Native Interface) qui est la technologie permettant de faire communiquer des codes Java et C / C++

2) Réduisez au maximum le nombre d’appel de Java vers C++ et encore plus de C++ vers Java, ces derniers rendent le code particulièrement complexe (voire incompréhensible) et ne devraient être utilisés qu’en cas de nécessité

3) Utilisez l’API C++ de Tango plutôt que l’API Java, nous aurions préféré effectuer la totalité du projet avec Java mais les limitations d’interopérabilité entre la caméra OpenCV (*CvCameraViewListener2*) et la caméra Tango (*TangoUpdateCallback* ou *OnTangoUpdateListener* pour les codes plus anciens) nous on fait perdre plusieurs jours et sont a priori difficilement contournable.

4) Utilisez ou inspirez-vous des exemples proposés par Google (lien à la fin du dossier), ils couvrent la plupart des situations que nous avons rencontré et sont dans l’ensemble bien réalisés (bien que parfois inconsistants avec des approches différentes pour effectuer la même action d’un exemple à un autre).

# Liens utiles :

* API Tango en C++ : <https://developers.google.com/tango/apis/c/>
* Exemples officiels en C++ : https://github.com/googlesamples/tango-examples-c/