École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2020-2021

Manuel développeur

Canne connectée pour aveugles

Encadrants
Gilles VENTURINI
gilles.venturini@etu.univ-tours.fr

Auteurs
Djawad M'DALLAH MARI
djawad.mdallah-mari@etu.univ-tours.fr

Université François-Rabelais, Tours

DII5 2020-2021

Version du 11 avril 2021

Table des matières

Introduction

Ce document fait partie d'un ensemble de livrables qui accompagne le projet de fin d'études "Canne connectée pour aveugles" réalisé en 2020-2021 à Polytech Tours par Djawad M'DALLAH-MARI.

C'est le manuel développeur qui vise toute personne souhaitant obtenir plus d'information sur la partie technique du système. Il détaille de manière précise chaque élément permettant de réaliser une tâche particulière dans le système.

Des notions en développement d'application, notamment en programmation événementielle, sont requises afin de mieux comprendre les éléments abordés dans ce document.

Installation des prérequis

Avant de pouvoir développer une application Android, il faut avoir installé au préalable quelques éléments.

2.1 Java Development Kit

Le Java Development Kit (JDK) permettra de compiler et déboguer le code. Il permettra également de l'exécuter grâce au JRE (Java Runtime Environment).

Pour l'installer il faut aller sur le site d'Oracle [1] et télécharger une version au-dessus de Java 8. Une fois téléchargé il suffit de suivre les instructions jusqu'à la fin de l'installation.

2.2 Android Studio et le SDK Android

Android Studio sera l'environnement de développement à utiliser pour les développements et le SDK Android permettra d'obtenir l'ensemble d'outils nécessaires pour le développement d'applications Android. Pour installer ces outils, télécharger l'installateur sur https://developer.android.com/studio. Une fois téléchargé il suffit de suivre les instructions jusqu'à la fin de l'installation en veillant à cocher l'installation du SDK Android.

2.3 Code source

Le code source du projet est disponible sur github [2]. Il suffit de le cloner ou télécharger le zip contenant tout le code. Ensuite, il faut **l'importer** sur Android Studio et lancer la compilation.

1. https://www.oracle.com/java/technologies/javase-downloads.html
2. https://github.com/Djawad-mdallahmari/PFE-ObjectDetection

Les fontionnalités implémentées

Dans cette partie, nous verrons comment sont implémentés les différentes fonctionnalité ajoutées à l'application.

3.1 Intégration du modèle

Le modèle, au format TensorFlow Lite (.tflite) est embarqué localement au sein de l'application. Ainsi, cela évite d'utiliser un modèle distant et donc d'être dans l'obligation d'avoir une connexion internet. Le modèle ainsi que son fichier label sont dans le répertoire `PFE-ObjectDetection/app/src/main/assets/`.

La création du détecteur se fait dans la classe `DetectorActivity` grâce à la méthode `TFLiteObjectDetectionAPIModel.create(...)`. Cette méthode de l'API TensorFlow Lite permet de créer un détecteur en prenant en paramètre, le contexte, le modèle, le fichier de label, ainsi que les paramètres spécifiant le modèle à savoir la taille des inputs et si le modèle est quantifié ou pas.

Pour utiliser ensuite le modèle, il suffit d'appeler la méthode `recognizeImage(croppedBitmap)` en lui passant l'image à analyser. Cette méthode est appelée pour chaque image du flux vidéo (grâce au listener qui déclenche la méthode `processImage()`). Cette méthode retourne une liste de résultats (`List<Detector.Recognition>`) qui va être traité avant que le résultat soit affiché/ennoncé à l'utilisateur.

3.2 Synthétiseur vocal

Le synthétiseur vocal utilise la classe `TextToSpeech`. L'objet est instancié lors de l'appellation de la méthode `onPreviewSizeChosen` (dans DetectorActivity) qui est appelé au début du programme. Lors de la création de l'objet on override la méthode `onInit` afin d'initialiser la langue du synthétiseur en anglais. En effet, le label des objets étant en anglais, il faut bien initialiser le synthétiseur, car la langue par défaut sera la langue du téléphone.

La lecture des objets identifiés est faite en appelant la méthode `speak()`. Dans notre cas, l'objet est cité qu'après avoir rempli les conditions suivantes :

- Il faut que l'objet soit visé -> `r.getLocation().contains(viseurReco.getLocation())`
- Il ne faut pas que le synthétiseur soit en train de cité un objet -> `!tts.isSpeaking()`
- Il ne faut pas que le synthétiseur soit désactiver (mute) -> `!isMuted`
- Si l'objet est le même que précédemment, on ne le cite qu'après un petit délai -> `readedText.equals(r.getTitle())`

```
if(r.getLocation().contains(viseurReco.getLocation())){ //Si le viseur est dans l'objet
    detecte && pas le meme
   if (readedText.equals(r.getTitle())){ // Si c'est le meme
     if(!tts.isSpeaking() && !isMuted){
```

```
            vibrator.vibrate(200); // Vibre
            tts.playSilentUtterance(3000,TextToSpeech.QUEUE_FLUSH,null);
            readedText = "";
         }
      }else{
         if(!tts.isSpeaking() && !isMuted){
            tts.speak(r.getTitle(), TextToSpeech.QUEUE_FLUSH, null, null); // Synthetiseur
                prononce le label de l'objet
            //vibrator.vibrate(200); // Vibre
            readedText = r.getTitle();
         }
      }
   }
```

3.3 Seuil de détection

Le réglage du seuil de détection se fait dans le volet déroulant. Les éléments graphiques ont donc été ajoutés au layout `tfe_od_layout_bottom_sheet.xml` dans lequel on ajoute deux boutons (plus et moins) ainsi qu'un affichage de la valeur, le tout encapsulé au sein de Layout (RelativeLayout et LinearLayout) afin d'être affiché correctement sur une ligne. Ces éléments sont ensuite récupérés dans la méthode `onCreate()` de l'activité `CameraActivity`. On définit l'activité même en tant que Listener des ces éléments en implémentant la classe `View.OnClickListener`. Ensuite dans la méthode `onClick`, appelée comme son nom l'indique lors d'un click sur les éléments, on vérifie l'élément graphique qui a été cliqué puis on fait le traitement.

```
   else if (v.getId() == R.id.plusSeuil) {
      String seuil = seuilTextView.getText().toString().trim();
      int numSeuil = Integer.parseInt(seuil);
      if (numSeuil >= 98) return;
      numSeuil+=2;
      System.out.println("numSeuil: "+numSeuil);
      seuilTextView.setText(String.valueOf(numSeuil));
      setNumSeuil(numSeuil);
    } else if (v.getId() == R.id.minusSeuil) {
      String seuil = seuilTextView.getText().toString().trim();
      int numSeuil = Integer.parseInt(seuil);
      if (numSeuil <= 2) {
        return;
      }
      numSeuil-=2;
      seuilTextView.setText(String.valueOf(numSeuil));
      setNumSeuil(numSeuil);
    }
```

Ici, on vérifie quel bouton a été appuyé puis on incrémente ou décrémente par pas de 2 le seuil de détection. Le seuil est ensuite mis à jour en appelant la méthode abstraite `setNumSeuil()` implémentée dans la classe `DetectorActivity`. La seule ligne intéressante de cette méthode est celle où on met à jour la variable globale du seuil de détection par la valeur reçu en paramètre `MINIMUM_CONFIDENCE_TF_OD_API = numSeuil/100f`. Ainsi, lors du prochain traitement, ce sera le nouveau seuil qui sera pris en compte.

3.4 Viseur virtuel

Pour la réalisation du viseur centrale, nous utilisons la classe `RectF` pour dessiner un rectangle qui représentera le viseur. Le rectangle est instancié avec son constructeur en indiquant les différentes coordonnées. Une fois initialisé, il suffit de rajouter une vérification en plus dans la liste des résultats retournés par le détecteur pour ne traiter l'objet que si il a été visé :
`if(r.getLocation().contains(viseurReco.getLocation())).`

3.5 Capteurs

Pour utiliser un capteur, il faut utiliser la classe `Sensor`. Ensuite, à l'aide d'un `SensorManager` on choisi le type de capteur souhaité `Sensor.TYPE_GEOMAGNETIC_ROTATION_VECTOR` par exemple. Ensuite il faut enregistrer le listener pour du capteur souhaité qui sera donc en charge de détecter les changements `sensorManager.registerListener(this,sensorGeomagnetic,10000)`. Ici le listener sera l'activité `CameraActivity` elle même donc elle devra implémenté la classe `SensorEventListener`.

Les changements seront ensuite détectés par la méthode `onSensorChanged` qu'il faut donc réimplémentée.

```
@Override
public void onSensorChanged(SensorEvent event) {
  if(event.sensor.getType() == Sensor.TYPE_ROTATION_VECTOR){
    gyroscopeTV.setText("gyroscope: x="+event.values[0]+" y="+event.values[1]+"
        z="+event.values[2]);
    gyroscopeTV.setTextColor(Color.WHITE);
  }
  if(event.sensor.getType() == Sensor.TYPE_PROXIMITY){
    proximityTV.setText("proximite: "+event.values[0]+" cm");
    proximityTV.setTextColor(Color.WHITE);
  }
  if (event.sensor.getType() == Sensor.TYPE_GEOMAGNETIC_ROTATION_VECTOR){
    geomagneticTV.setText("geomagnetique: x="+event.values[0]+" y="+event.values[1]+"
        z="+event.values[2]);
    geomagneticTV.setTextColor(Color.WHITE);
  }
}
```

Ici ,on vérifie d'abord le type de capteur pour savoir lequel a changé, ensuite, on met à jour les valeurs. Les valeurs sont, à ce jour, affichées directement à l'écran, mais pourrait être traité directement pour implémenter une fonctionnalité utilisant ces capteurs.

Le capteur de localisation est quant à lui est implémenté différement. Il faut utiliser la classe `LocationManager` et demander les permissions avant de pouvoir continuer. Ceci est fait grâce à la méthode `requestPermissionLocation()` qui est appelée que si on n'a pas encore les permissions.

```
if(hasPermissionLocation()){
      locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
      locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, this);
      locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, this);
    }else{
      LOGGER.i("Request permission");
      requestPermissionLocation();
    }
```

Ici, on utilise deux capteurs, bien qu'un seul suffit pour avoir la géolocalisation. Le premier se basant sur les coordonnées GPS tandis que l'autre se basant sur le réseau où l'utilisateur est connecté. Il faut ensuite implémenter le listener `LocationListener` afin d'être notifié des changements de position géographique.

Documentation

Dans ce manuel, nous nous sommes contenté de commenter les parties du code qui réalise les fonctionnalités implémentées. Le reste du code sera facilement compréhensible par le développeur. Nous invitons en cas de difficulté à se référer à la documentation de TensorFlow Lite.

Table des figures

Liste des tableaux

Canne connectée pour aveugles

Département Informatique
5e année
2020-2021

Manuel développeur

Résumé : Manuel développeur canne connecée pour aveugles

Mots clefs :

Abstract:

Keywords:

Encadrants
Gilles VENTURINI
gilles.venturini@etu.univ-tours.fr

Auteurs
Djawad M'DALLAH MARI
djawad.mdallah-mari@etu.univ-tours.fr

Université François-Rabelais, Tours

DII5 2020-2021