Manuel d'installation

Implémentation d'un système d'aide à la vision sur des lunettes de réalité virtuelle

- **Client :** Olivier BODINI <olivier.bodini@univ-paris13.fr>

- **Équipe de suivi :**

  Thierry HAMON <thierry.hamon@univ-paris13.fr>

  Sophie TOULOUSE <sophie.toulouse@lipn.univ-paris13.fr>

- **Groupe :**

  Mohamed Ali YACOUBI <mohamedali.yacoubi@edu.univ-paris13.fr>

  Flavien HAMELIN <flavien.hamelin@edu.univ-paris13.fr>

  Safa KASSOUS <safa.kassous@edu.univ-paris13.fr>

  Farah CHERIF <farah.cherif1@edu.univ-paris13.fr>

  Saad AMMARI <saad.ammari@edu.univ-paris13.fr>

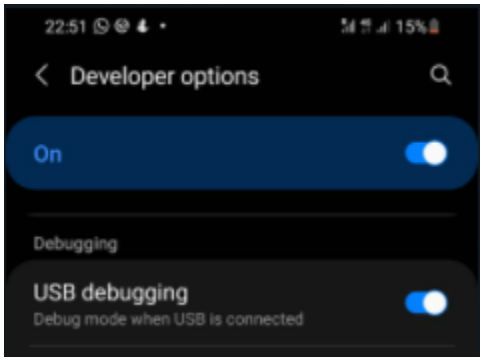
25/04/2022

Table des matières

I) Mise en place

Nous avons utilisé Java 16.0.2 pour développer ce projet. Vous pouvez télécharger Java en cliquant sur ce lien : Java Downloads for Linux. Pour mettre en marche l'application, il faut installer l'IDE Android Studio. Vous pouvez le télécharger en cliquant sur le lien suivant: Download Android Studio and SDK tools. Une fois installé, vous pouvez brancher les lunettes (ou votre smartphone) au PC avec un câble USB, charger le projet sur Android Studio et cliquer sur "Run" dans Android Studio. Android Studio va compiler le programme, installer l'application sur l'appareil et ouvrir l'application automatiquement.

Remarque 1: Pour tester l'application sur smartphone, il faut activer l'option "USB debugging" dans les options développeur du smartphone comme le montre la figure suivante :



Remarque 2: Pour signaler qu'un point est vu lors du test lancé par la génération d'une nouvelle cartographie il faut utiliser le bouton : "Volume bas" de votre appareil (lunettes ou smartphone).

II) Les principales classes

- **MainActivity**: c'est l'activité qui est responsable de la gestion de l'interface d'accueil, c'est ici où l'application va lire une cartographie lorsqu'on clique sur "Choisir une cartographie".
- **DisplayPoint**: c'est l'activité responsable de la génération des points lumineux.

- **CartographyActivity**: c'est la classe responsable de l'affichage des points avec la fonction "**start**" et de contrôler les réponses de l'utilisateur avec la fonction "**dispatchKeyEvent**".
- **CartographyDraw**: c'est l'activité responsable d'utiliser la classe ConvexHullAlgorithm pour calculer l'enveloppe convexe des points non vus et de la dessiner.

III) Les principales fonctions

- Classe CartographyDraw
    - Fonction **getConvexHull** : permet de retourner une enveloppe convexe à partir d'une liste de DisplayPoint.
    - Fonction **computeCentroid**: permet de calculer le barycentre des points non vus pour imposer une condition d'appartenance à l'enveloppe convexe.
- Classe DisplayPoint
    - **Constructeur** : permet de générer un point de paramètres aléatoires en phase aléatoire et d'explorer un voisinage de Von Neumann dans une phase non aléatoire.
- CartographyActivity
    - Fonction **OnCreate** : permet de déclencher la fonction Start et d'initialiser les variables globales.
    - Fonction **Save_data** : permet d'enregistrer une cartographie générée par un test de vision.
    - Fonction **Start :** permet de commencer le test de vision qui dure 20 minutes (10 minutes pour chaque oeil).
    - Fonction **onKeyDown**: permet de ne pas afficher la barre de volume lorsque l'utilisateur clique sur le bouton bas du volume.
    - Fonction **dispatchKeyEvent:** permet de capturer les cliques de l'utilisateur et de réaliser une suite logique à ses actions.
- Classe CameraActivity
    - Fonction **onCreate:** permet de décaler le retour de la caméra dans un coin.

IV) Affichage des points lumineux

La fonction **start()** est la fonction principale appelée pour afficher les points lumineux. Il y a deux blocs dans cette fonction, le premier bloc est utilisé si la variable booléenne “randomPhase” est à vrai, dans ce cas l’algorithme va appeler la classe DisplayPoint pour générer des points aléatoirement. Sinon, si “randomPhase” est à faux, cela veut dire qu’un point n’a pas été vu, donc l’algorithme va s’intéresser à une zone aveugle bien déterminée.

Nous faisons deux tests, chaque test prend 10 minutes, la figure ci-dessous présente, le début du code du test de l’oeil droit.

```
if(System.currentTimeMillis()-t0>10*60*1000){
    save_data();
    workerThread();//display message
    Timer timer1 = new Timer();
    long t1 = System.currentTimeMillis();
    randomPhase=true;
    four_neighbours_checking = false;
    direction_history.clear();
    ISCLICKED=false;
    ISSHOWED=false;
    treatedPointsB.clear();

    //we choose to always visit the north at first
    visitingNorth=true;
    next_direction="north";
    direction_history.add("north");
    if(!direction_history.contains("south"))
        direction_history.add("south");
    //traitement de left eye test
    isRightEye=false;
    timer1.scheduleAtFixedRate(() → {
```

V) Sauvegarde de la cartographie

Nous sauvegardons les cartographies sous format Json dans un fichier texte qui a pour nom la date de génération de la cartographie avec le suffixe “Gauche” pour l’oeil gauche, et “Droit” pour l’oeil droit. Les cartographies sauvegardées sont sous le répertoire “Downloads”. Après avoir généré une cartographie vous pouvez donc aller la chercher dans le répertoire “Downloads” via l’option de l’interface d’accueil : “Choisir

une cartographie". La figure ci-dessous montre le programme responsable à la sauvegarde d'une cartographie.

```
/**
 * Permet d'enregister un objet dans un fichier
 *
 * @param serializedObject L'objet qu'on va enregistrer dans un fichier
 * @param filename le nom du fichier dans lequel on va enregistrer l'objet
 */
private void writeObjectToFile( String serializedObject, String filename) {

    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        if (Build.VERSION.SDK_INT >= 23) {
            if (checkPermission()) {
                File sdcard = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS)
                File dir = new File( pathname: sdcard.getAbsolutePath() + "/DMLA/");
                dir.mkdir();
                File file = new File(dir,  child: filename+".txt");
                FileOutputStream os = null;
                try {
                    os = new FileOutputStream(file);
                    os.write(serializedObject.getBytes(StandardCharsets.UTF_8));
```

VI) Lecture d'une cartographie

Dans la classe MainActivity, plus précisément lorsque l'utilisateur clique sur le bouton : "Choisir une cartographie", l'algorithme de recherche et d'importation de la cartographie générée précédemment va se déclencher. La figure ci-dessous présente le début de l'algorithme en question.

```
case R.id.buttonPickCarto:
    new AlertDialog.Builder( context: this)
            .setTitle("Pas de cartographie sélectionnée")
            .setMessage("Choisir une cartographie ?")
            .setPositiveButton("Rechercher",
                    new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            //intent[0] = new Intent(Intent.ACTION_OPEN_DOCUMENT);
                            intent[0] = new Intent(Intent.ACTION_GET_CONTENT);
                            intent[0].addCategory(Intent.CATEGORY_OPENABLE);
```

VII) Affichage d'une cartographie

Pour afficher une cartographie, nous lisons le fichier d'extension ".txt" d'une cartographie, nous appelons l'algorithme qui génère l'enveloppe convexe et nous affichons le résultat. Le bouton "Afficher une cartographie" vous permet d'afficher la toute dernière cartographie générée (la plus récente dans le répertoire de sauvegarde). La figure ci-dessous présente l'algorithme d'affichage.

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    int i=0;
    wallpath.reset();
    if(getConvexHull(nvpr).size()>0){


    Point centroid = computeCentroid(getConvexHull(nvpr));
    for (Point p:getConvexHull(nvpr)) {
        DisplayPoint curr= new DisplayPoint(p);
        for (DisplayPoint dp: nvpr) {
            //canvas.drawCircle(dp.getX(),dp.getY(),dp.getRadius(),seeCircle);
            if (curr.equals(dp)){
                i++;
                if (i==1){
                    wallpath.moveTo(dp.getX(), dp.getY());
                    continue;
                }
                // used for first point
                Point testPoint = new Point(dp.getX(),dp.getY());
                if(computeDistance(testPoint,centroid)<1.25*averageDistance(centroid,getConvexHull(nvpr))){
                    wallpath.lineTo(dp.getX(), dp.getY());
                }
```

VIII) Test automatique

Nous avons fait un robot qui, à partir d'une zone aveugle définie, teste notre algorithme lorsque vous choisissez l'option : "Nouvelle cartographie". Pour faire fonctionner le robot il suffit de changer la valeur de la variable globale "robotTest" à true, dans la classe cartographyActivity et réinstaller l'application comme le montre la figure ci-dessous.

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    View decorView = getWindow().getDecorView();
    // Hide both the navigation bar and the status bar.
    // SYSTEM_UI_FLAG_FULLSCREEN is only available on Android 4.1 and higher, but as
    // a general rule, you should design your app to hide the status bar whenever you
    // hide the navigation bar.
    ISCLICKED=false;
    ISSHOWED=false;
    robotTest=false;
```

IX) Ressources

L'algorithme de l'enveloppe convexe: GitHub - bkiers/GrahamScan: A Java implementation of the Graham Scan algorithm to find the convex hull of a set of points.

Présentation globale (vidéo): https://www.youtube.com/watch?v=RcWlptuHFwA

Test humain (vidéo): https://www.youtube.com/watch?v=37LgwVWGMq8

Test robot 20 mins (vidéo): https://www.youtube.com/watch?v=BIeOdJy6TIE