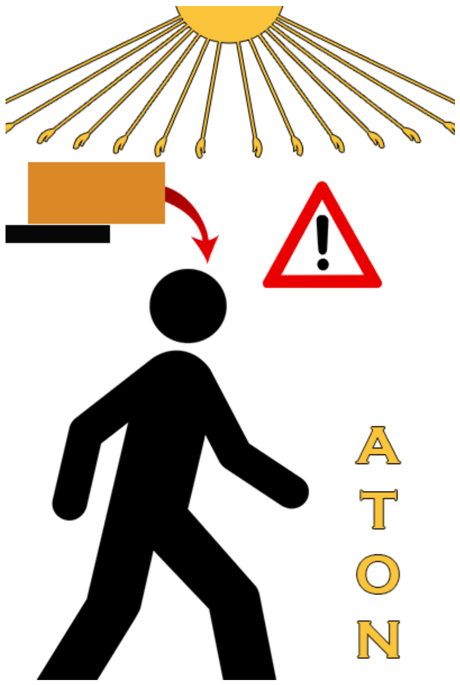


DEVELOPER MANUAL

ATON (Alert about Threaten Objects Near you)

Hugo BALTZ

8/19/2016

Table of Contents

Important notes:

This application has been developed on Android Studio 2.1.1.

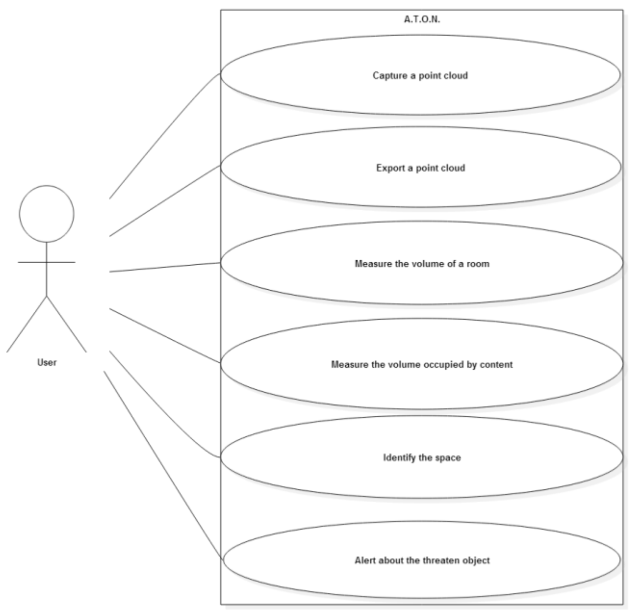
The minimum SDK version to launch the application is 17, and the target SDK is 19.

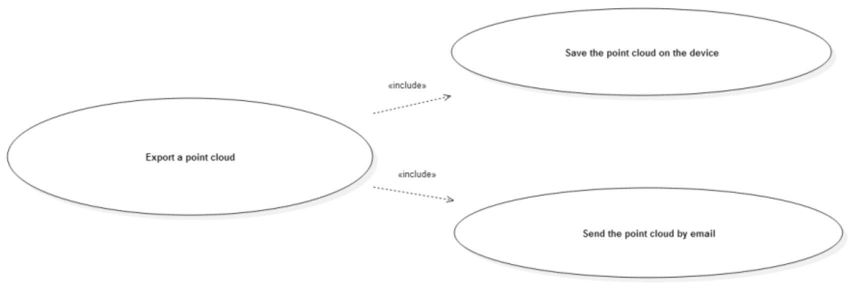
This application uses the library tango support, Tango SDK Qianru Java, and the library rajawali 3D V1.0.325.

My report explains the theoretical solution, in this documentation I don't explain in detailes the methods, only how I implemented them.

Introduction:

This application acquires a point cloud, stores the point cloud on the device, can delete the point cloud on the device, calculates the volume of a room (using the point cloud of the room), in the future it will calculate the volume occupied of the room, will determine the type of the room and if an object is dangerous in case of an earthquake.


A.T.O.N.
Capture a point cloud
Export a point cloud
Measure the volume of a room
Measure the volume occupied by content
Identify the space
Alert about the threaten object
User


Export a point cloud
«include»
Save the point cloud on the device
«include»
Send the point cloud by email

I. Code structure:

You will find the detail of each class in the Javadoc of this project, If you have question you can contact me at hugo.baltz@gmail.com.

The code is divided in eight parts; I explain in this paragraph the interest of each of this party.

I.1 Rajawali:

It is a library of the project tango, I add it in the project, because Android studio did not succeed to build my project with this library on the dependencies. I will not develop how this part works. I use this library to manage the point cloud and the augmented reality.

I.2 MainActivity:

It is the activity that initialize the project: it starts the AR (Augmented Reality), setups the extrinsic, creates the menu, initializes the listener on the button of the menu.

I.3 Renderer:

This folder contains the classes that manage the point clouds, and specifically all the visual aspect in the AR but also the actions associates to each buttons of the menu (see the user manual).

I.3.1 Materials:

This class contains the materials that define the aspect of the point clouds, specifically the color of the point clouds.

I.3.2 PointCloudARRender:

This class extends the renderer TangoRajawaliRenderer, it manages everything that it dispaly on the screen, such as the point cloud created by the device, the point cloud collected by the application, and it manages the action links to the menu.

I.3.3 PointCollection:

This class extends the class Object3D (org.rajawali3d.Object3D), it corresponds to the point clouds. The fields of this class are a FloatBuffer: the point cloud, an integer mMaxNumberOfVertices which defines the maximum of points in the point cloud, and an integer count which is the number of points in the point cloud.

I.4 Utilities:

This folder contains the classes that are useful to other classes.

I.4.1 PointCloudExporter:

This class manages the exportation of a file, it is this class that write on the internal memory the .xyz.

I.4.2 PointCloudManger:

This class manages the point cloud, it fills the current points (the ones that are create by the device) and it fills the collected points (the ones store on the device).

I.4.3 PointCloudVolumeCalculator:

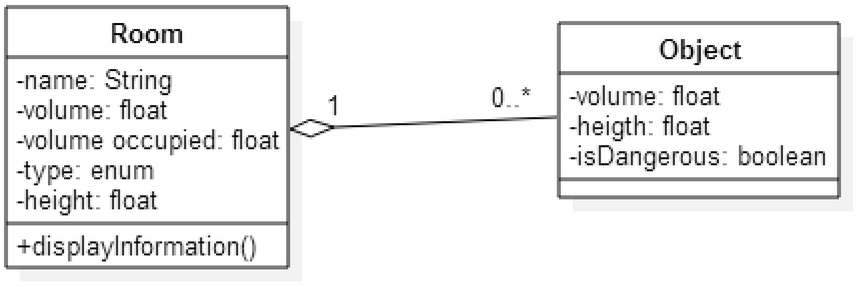
This class manages the calculation of the volume of a room and displays the results on the screen.

I.4.4 Various:

This class contains several useful functions that can be use by the others classes of the applications.

I.5 Object:

This folder contains the classes Room and Object:



I.6 Polygon:

This folder contains the classes useful to create a polygon.

I.6.1 Point:

This class has two fields X and Y (two floats).

I.6.2 Line:

This class uses the class point; a line is formed by two points.

I.6.3 Polygon:

This class uses the line class; a polygon is defined by his sides which are line.

I.6.4 Angle:

This class is useful for the JarvisMarch algorithm which is an algorithm to calculate the convex hull of a finite number of points.

I.7 Provider:

This folder contains only the class MailProvider.

This class is useful to send an email with an attachment which is on the internal storage of the device.

I.8 Hull:

This folder contains the class JarvisMarch.

This class implements the algorithm of Jarvis' march which is an algorithm that calculate the convex hull of a finite number of points.

II. How the application works:

In this section, I will explain how the different functionalities of the application work.

II.1 Capture a point cloud:

To capture the point cloud, the application passes by three phases: first it creates a point cloud, then it displays this point cloud in the screen and at the end it cans store a point cloud. So the application creates the point cloud of a room.

II.1.1 Create a point cloud:

To create the point cloud, we use the project tango library and the rajawali library. In the onCreate() of MainActivity, we see that we first initialize the rendrer:

```
glView = new TangoRajawaliView(this);
renderer = new PointCloudARRenderer(this);
glView.setSurfaceRenderer(renderer);
glView.setOnTouchListener(this);
setContentView(R.layout.activity_main);
RelativeLayout wrapper = (RelativeLayout) findViewById(R.id.wrapper_view);
tango = new Tango(this);
startActivityForResult(Tango.getRequestPermissionIntent(Tango.PERMISSIONTYPE_MOTION_TR
ACKING), Tango.TANGO_INTENT_ACTIVITYCODE);
wrapper.addView(glView);
```

The wrapper has the size of the screen.

The function startActivityForResult() verifies if the application as the permission for the motion tracking. We can see in the onActivityResult() that if the application doesn't have the permission to the motion tracking then it displays a message which says that the motion tracking permission is required. Else, it launch the function startAugmentedReality():

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // Result on the permmision of the motion tracking:
    if (requestCode == Tango.TANGO_INTENT_ACTIVITYCODE) {
        if (resultCode == RESULT_CANCELED) {
            Toast.makeText(this, "Motion Tracking Permissions Required!",
Toast.LENGTH_SHORT).show();
            finish();
        } else {
            startAugmentedReality();
            isPermissionGranted = true;
        }}}
```

The function startAugmentedReality() initializes the connection to the camera if it doesn't exist. So if the application is not connected to the camera, the function initializes the camera especially it allows the depth perception:

```
glView.connectToTangoCamera(tango, TangoCameraIntrinsics.TANGO_CAMERA_COLOR);
TangoConfig config = tango.getConfig(TangoConfig.CONFIG_TYPE_DEFAULT);
config.putBoolean(TangoConfig.KEY_BOOLEAN_LOWLATENCYIMUINTEGRATION, true);
config.putBoolean(TangoConfig.KEY_BOOLEAN_DEPTH, true);// Allow the depth perception
tango.connect(config);
```

Then it sets the listener on the frame pairs. This listener has a method onXyzIjAvaible(), this method is called when we have XYZ information, so when we have a point cloud. This is how the point cloud is created using the project tango library.

II.1.2 Display the point cloud:

Now that we have a point cloud, we want to display it on the screen, for that still in the onXyzIjAvaible() of listener, we update the XyzIjData of the point cloud manager (for recall it is the class that manage the display of the point clouds on the screen):

```
pointCloudManager.updateXyzIjData(xyzIj, cloudPose);
```

In startAugmentedReality(), we link the pointCloudManager to the renderer:

```
// Associate the point cloud manager to the renderer:
pointCloudManager = new
PointCloudManager(tango.getCameraIntrinsics(TangoCameraIntrinsics.TANGO_CAMERA_COLOR))
;
renderer.setPointCloudManager(pointCloudManager);
```

Let now see how the renderer works to display this point cloud, in the onRenderer() of PointCloudARRenderer, if the point cloud manager has new points then we fill the variable currentPoints of the PointCloudARRender:

```
if (pointCloudManager != null && pointCloudManager.hasNewPoints()) {
    Pose pose =
mScenePoseCalculator.toOpenGLPointCloudPose(pointCloudManager.getDevicePoseAtCloudTime
());
    pointCloudManager.fillCurrentPoints(currentPoints, pose);
}
```

The function fillCurrentPoints() set the FloatBuffer, the position and the orientation of the PointCollection currentPoints:

```
public synchronized void fillCurrentPoints(Points currentPoints, Pose pose) {
    currentPoints.updatePoints(xyzIjData.xyzCount, xyzIjData.xyz);
    currentPoints.setPosition(pose.getPosition());
    currentPoints.setOrientation(pose.getOrientation());
    }
```

In the initScene() of the renderer, the current points are add to the scene, so this is are there are display on the screen. Furthermore, in this function we set the color of the current point cloud to green:

```
currentPoints.setMaterial(Materials.getGreenPointCloudMaterial());
getCurrentScene().addChild(currentPoints);
```

II.1.3 Store a point cloud:

We capture the point by touching the screen it defines in the onTouch() of the MainActvity:

```
if (motionEvent.getAction() == MotionEvent.ACTION_UP) {
    renderer.capturePoints();
}
```

This function in the renderer, set the Boolean collectPoints to true

```
public void capturePoints() {
    collectPoints = true;
}
```

Then in onRender(), we add the Xyz information of point cloud manager to the collected points:

```
if (collectPoints) {
    collectPoints = false;
    pointCloudManager.fillCollectedPoints(collectedPoints, pose);
}
```

For the current points, the previous points are not save, in this case we add the point to the point collection, so we can create a point cloud of a room.

To display the collected points on the screen, it is like the current point except this time their color is blue.

II.2 Export a point cloud:

This functionality regroups two functionalities: save the point cloud on the device and send the point cloud by email.

II.2.1 Save the point cloud on the device:

I decide in the interest of saving time to save the file on the internal memory of the device in the form of a file .xyz and not in a database.

To save a point cloud, the user has to touch the button "Export Pointcloud" on the menu. In onOptionsItemSelected() of the MainActivity, let see what action is associate to this button:

```
case R.id.activity_main_menu_export_pointcloud:
    renderer.exportPointCloud(this);
    return true;
```

In PointCloudARRenderer, the function exportPointCloud() creates a dialog where the user has to enter the name of the room:

```
final Dialog dialog = new Dialog(mainActivity);
dialog.setContentView(R.layout.dialog_export);
dialog.setTitle(mainActivity.getString(R.string.titleName));

Button dialogButton = (Button) dialog.findViewById(R.id.dialogButtonOK);
final EditText nameRoom = (EditText) dialog.findViewById(R.id.nameRoom);
```

Then we set the listener on the button of the dialog, this listener when he is activated, create a new PointCloudExporter and launch the function export() of this exporter:

```
dialogButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        String name = nameRoom.getText().toString();

        if(!name.equals("")) {
            Log.d("sizeCP", "" + collectedPoints.getCount());
            PointCloudExporter exporter = new PointCloudExporter(mainActivity, name,
collectedPoints);
            exporter.export();

            dialog.dismiss();
        } else {
            Various.makeToast(mContext,mainActivity.getString(R.string.noName));
        }
    }
});
```

Let see how the exporter works, the function export() of the exporter recovers the FloatBuffer of the point collection, then it calls the function createFile() placed in the class Various.

```
PointCollection pointCollection = params[0];
FloatBuffer floatBuffer = pointCollection.getBuffer();
int size = pointCollection.getCount();

Various.createFile(context, roomName, floatBuffer, size);
```

When the exportation is over, it displays a message on the screen:

```
protected void onPostExecute(Void aVoid) {
    super.onPostExecute(aVoid);

    Various.makeToast(context,"Point cloud exported!");
}
```

The function createFile() creates a file in the internal memory with the form pointcloud-“name of the room”.xyz. The file is create on the cache of the application:

```
String fileName = String.format("pointcloud-%s.xyz", roomName);
File f = new File(context.getCacheDir() + File.separator);
final File file = new File(f, fileName);
filePath = file.getPath();
```

Then we create a file using the FileOutputSream:

```
FileOutputStream fos = new FileOutputStream(cacheFile);
OutputStreamWriter osw = new OutputStreamWriter(fos, "UTF8");
PrintWriter pw = new PrintWriter(osw);
```

We loop on the float buffer and we write for each point a line of XYZ:

```
for (int i = 0; i < size; i++) {
    String row = String.valueOf(floatBuffer.get()) + " " +
String.valueOf(floatBuffer.get()) + " " + String.valueOf(floatBuffer.get());
    pw.println(row);
}

pw.flush();
pw.close();
```

II.2.2 Send the point cloud by email;

Now that we have the .xyz of the point cloud, we would like to see it on a computer, so we would to send this file by email.

Like the file is on the internal memory and that we want to use an extern application (Gmail, drive…) to send the email with a point cloud in the attachment, we have to create a provider and allow this provider to access to the file of our application.

The MailProvider extends ContentProvider, this provider defines how the uri has to be define and creates the ParcelFileDescriptor:

```
// Check incoming Uri against the matcher
switch (uriMatcher.match(uri)) {
    // If it returns 1 - then it matches the Uri defined in onCreate
    case 1:
        // The desired file name is specified by the last segment of the path
        // Take this and build the path to the file
        String fileLocation = getContext().getCacheDir() + File.separator +
uri.getLastPathSegment();

        // Create & return a ParcelFileDescriptor pointing to the file
        // Note: I don't care what mode they ask for - they're only getting
        // read only
        ParcelFileDescriptor pfd = ParcelFileDescriptor.open(new File(fileLocation),
ParcelFileDescriptor.MODE_READ_ONLY);
        return pfd;

    // Otherwise unrecognised Uri
    default:
        Log.v(LOG_TAG, "Unsupported uri: '" + uri + "'.");
        throw new FileNotFoundException("Unsupported uri: " + uri.toString());
}
```

I think the comments as sufficient to understand this part of the code.

In the manifest, you have to allow the provider to export the content of the application:

```
<provider android:name="com.example.hbaltz.aton.provider.MailProvider"
android:authorities="com.example.hbaltz.aton"  android:exported="true"
android:enabled="true"/>
```

To launch this export, the user, click on the menu's button "Send PointCloud", in onOptionsItemSelected() of the MainActivity:

```
case R.id.activity_main_menu_send_pointcloud:
    renderer.sendPointCloud(this);
    return true;
```

The function sendPointCloud() of the renderer, first display a dialog where the different file XYZ are listed, the user has to chooses one, when he chooses one of the file, he has to enter an email address:

```
AlertDialog.Builder builder = new AlertDialog.Builder(mainActivity);
builder.setTitle(mainActivity.getString(R.string.chooseRoom));
ArrayList<String> listTemp = Various.recoverListOfFiles(mainActivity);
final CharSequence[] nameRoom = Various.ArrayList2CharSeq(listTemp);
```

I think the code is easily comprehensible, the important part of this code is:

```
mainActivity.startActivity(getSendEmailIntent(mail,
        mainActivity.getString(R.string.subj_email) + " " + nameRoom[pos],
        mainActivity.getString(R.string.body_email) + " " + nameRoom[pos],
        filename));
```

This here that we create and launch the activity to send the xyz file by email.

The function getSendEmailIntent() creates the Intent that contains all the information in the email like the recipient, the subject, the body, the attachment. Then it uses the MailProvider to has the right to export a file from the internal memory:

```
//Add the attachment by specifying a reference to our custom MailProvider
//and the specific file of interest
emailIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse("content://" +
MailProvider.AUTHORITY + "/" + fileName));
```

II.3 Measure the volume of a room:

The calculation of the volume of the room passes by three steps: detect the ceiling and the floor of the point cloud, calculate the convex hull of the ceiling and the area of this convex hull, multiply this area if the difference between the y of the ceiling and the y of the floor (which is the height of the room).

II.3.1 Detect the ceiling and the floor of the point cloud:

The functions useful to detect the ceiling and the floor are methods of the Various class. I only explain here how to detect the ceiling, the detection of the floor uses a similar technique.

The function detectCelling() returns the point cloud of the ceiling (it is an ArrayList of triplet that we convert later on a floatBuffer to export him). To detect the ceiling, we find the maximal y (= yMax) of the point cloud and all the point that are at a distance to yMax inferior at a limit are a part of the ceiling. To find the yMax, we have the function findYMax() that is a classical function to find a maximum.

So as I explain it, now that we have the yMax all the point with a y sufficiently close to yMax are add to the ceiling:

```
if ((yMax - y) <= accuracy) {
    pt = new float[3];
    pt[0] = x;
    pt[1] = y;
    pt[2] = z;

    ceiling.add(pt);
}
```

II.3.2 Calculate the convex hull of the ceiling and his area:

To calculate the convex hull, we need geometry, this is why we have created the class on polygon which are geometries (for more details you can read the Javadoc).

We consider that the ceiling is a 2D plan y = constant, so we only work on the X and Z of the points.

We use the algorithm of the Jarvis March to find the convex hull, it is an algorithm that create the convex hull of a finite number of points. (if you want more theoretical information about this algorithm see my report).

The class JarvisMarch is the simple implementation of the algorithm, so it doesn't present any difficulty to be understand if you have understood the Jarvis march algorithm. The only particularities are that my start point is the "lowest" point in the plan (the one if the minimal z).

Now that we have the convex hull which is a polygon, we can calculate his area. In the class Polygon, there is a methods calculateArea(), it is a classical method to calculate the area of a polygon, you can find an algorithm there.

II.3.3 Calculate the volume of the room:

To calculate the volume of a room, the user clicks on the button "Calculate the volume", in onOptiomsItemSelected() of the MainActivity:

```
case R.id.activity_main_menu_calculate_volume:
    renderer.calculateVolumeRoom(this);
    return true;
```

In the renderer the function calculateVolumeRoom() displays a list of the room in the device's memory like when we want to send the xyz by email (see II.2.2). Then it creates a volume calculator and launch the calculation:

```
PointCloudVolumeCalculator calculator = new PointCloudVolumeCalculator(mainActivity,
(String)nameRoom[which]);
calculator.calculate();
```

The PointCloudVolumeCalculator uses the name and the point cloud to creates an object room, when we instantiate the class room, then the volume is automatically calculated:

```
FloatBuffer FBImp = Various.readFromFile(context,fileName);
room = new Room(roomName,FBImp);
```

In the Class room's constructor Room(String name, FloatBuffer pc), first the ceiling and the floor are detected. Then the y of the ceiling and the y of the floor are determined (it is the median of the y in the point cloud of the ceiling and of the floor):

```
ArrayList<float[]> ceiling =
Various.detectCelling(pointCloud,pointCloud.position()/3,1f);
ArrayList<float[]> floor = Various.detectFloor(pointCloud,pointCloud.position()/3,1f);

float yCeil = Various.findYMedian(ceiling);
float yFloor = Various.findYMedian(floor);
this.height = yCeil - yFloor;
```

Then the constructor uses the Jarvis march to determine the convex hull of the ceiling and his area:

```
JarvisMarch jarvisMarch = new JarvisMarch();
Polygon convCeiling = jarvisMarch.convexHull(ceiling);
```

Then we calculate the volume of the room:

```
this.volume = height * convCeiling.getArea();
```

III. Notes:

I think that the rest of the code is easily comprehensible, if I am wrong, please contact me by e-mail at hugo.baltz@gmail.com,.