ArmDroid – an Android app for teleoperating the KUKA youBot mobile manipulation platform

Matteo Morelli

May 25, 2012

Contents

1 Introduction

This section introduces ArmDroid, an Android application (app) developed as final project for the course "Android Framework" taught at the TeCIP Institute, Scuola Superiore Sant'Anna (Pisa, Italy), from 11/11 to 12/02, and valid for 3 ECTS-credits[1]. ArmDroid falls in the category 2 of the proposed projects: "Sensor Network and Controller" [3].

1.1 What ArmDroid is and how it works

ArmDroid allows to teleoperate the KUKA youBot mobile manipulation platform [1] via an Android-powered device, such as a tablet or a smartphone.

As detailed in [5], the KUKA youBot consists of two main parts: the *omnidirectional mobile platform*, whose motions are described by red arrows in figure 1(a), which consists of the robot chassis, 4 mecanum wheels, motors, power and an onboard PC; and the *robot arm*, which has 5 degrees of freedom (DOFs), whose motions around their axes are shown as blue arrows in figure 1(a), and can be controlled by the onboard PC if connected to the mobile platform.

Figure 1(b) depicts the way the teleoperation is performed by the user. The ArmDroid application provides a (simplified) "Teach-Pendant[2]"-like activity with edit texts and push buttons to pilot the robot arm and to save joint configurations in a local database; it also provides a "Wii-mote"-like activity that relies on sensors (accelerometer and magnetic field) to drive the mobile platform. Motion commands are periodically acquired by a (non-hard-real-time) task, and get transmitted via UDP to a server node written in Python language. The server node is distributed as a package in the ROS framework [4], and runs on a remote PC; it may operate the actual system (if the remote PC is the youBot's onboard PC), or simulate the robot in the Gazebo simulator [2].

ArmDroid must not be intended as a feature-rich application for robot teleoperation. Instead, it is designed to be a simple application that does work. In this context, *simple* means that simplification assumptions have been made about the application logic according to the objectives of the project, that, in the end, is about a course on the Android framework, not robotics or control. On the other hand, the application should also *do something interesting* (entertaining?). In this sense, the time that a full-fledged user interface and the application of sophisticated techniques of software engineering would have required to be, respectively, developed and applied, has been "sacrificed" in favor of the implementation of concrete (even if basic) teleoperation functionalities.

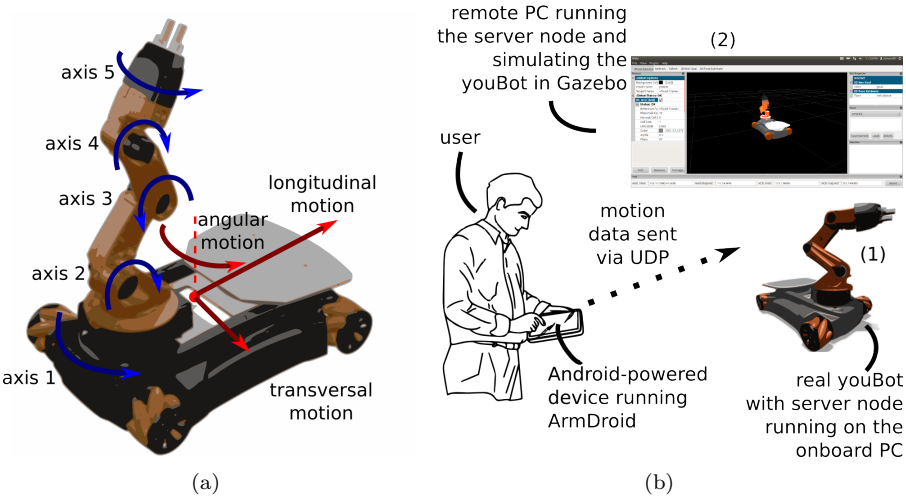
1.2 Hardware/Software pre-requirements

ArmDroid has been developed in Eclipse, version 3.7.1, with Android Development Toolkit version 16.0.1.v201112150204-238534.

The application has been tested on an actual hardware device, a Samsung Galaxy S GT-I9000 with firmware version 2.3.5 (GINGERBREAD.NEJVK). The "Teach-Pendant"-like activity, which does not rely on sensors, has been also tested on an AVD whose system image target is Platform 2.3.3 - API Level 10.

The mobile manipulation platform has been operated through ROS Electric.

[1] `http://en.wikipedia.org/wiki/European_Credit_Transfer_and_Accumulation_System`
[2] `http://en.wikipedia.org/wiki/File:Teach_pendant.JPG`



(a) (b)

Figure 1: (a): Overview of the motions of the KUKA youBot omni-directional base (red arrows), and of the its arm (blue arrows). (b): Birds-eye view of the ArmDroid design; ArmDroid periodically sends motion data to a remote server node, which may either operate the real youBot (1) or simulate it in Gazebo (2).

1.3 License, right-of-use and citation

ArmDroid is copyright © by Matteo Morelli, 2012; it is free-software, released under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License[3], or any later version.

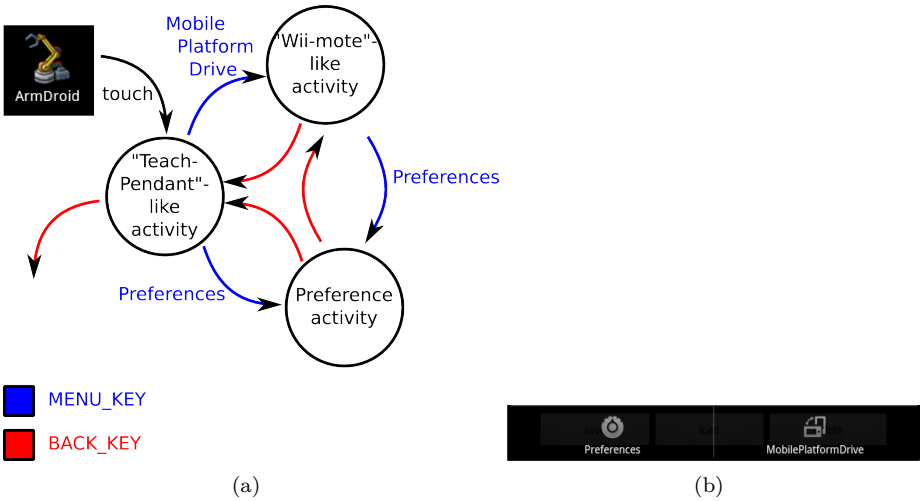
To cite ArmDroid, the following BibTeX entry must be used:

```
@Techreport{morelli12:armdroid,
    title       = "Arm{D}roid -- an {A}ndroid app for
                     teleoperating the {KUKA} you{B}ot mobile
                       manipulation platform",
    author      = "Matteo Morelli",
    institution = "ReTiS Lab., Scuola Superiore di Studi
                     Universitari e di Perfezionamento Sant'Anna",
    year        = "2012"
}
```

2 Getting started with ArmDroid

This section describes how to get started with ArmDroid. First, the application life-cycle is described, then the available settings and preferences are listed. The section ends by giving an example of how a teleoperation task can be performed, for both the arm and the mobile base of he KUKA youBot.

[3]http://www.gnu.org/licenses/gpl-3.0.txt



(a) (b)

Figure 2: (a): ArmDroid application life-cycle. (b): ArmDroid options menu.

2.1 Overview of the application life-cycle

ArmDroid is started by (locating and) touching the ArmDroid launcher icon. This action starts the `TeachPendantActivity`, i.e., the activity that implements the "Teach-Pendant"-like mode of the application. From this activity, after having pressed the `MENU_KEY` on the device, the user may either launch the `MobilePlatformDriveActivity`, i.e., the activity that realizes the "Wii-mote"-like mode of the application, or access the preferences, to change the application settings.

In order to return back to the `TeachPendantActivity`, the user must use the `BACK_KEY` on the device, since the `MENU_KEY` does not work when the prefe-rece screen is displayed, and it only allows to access the preferences when the `MobilePlatformDriveActivity` is running. The action of pressing the `BACK_KEY` when the `TeachPendantActivity` is running will exit the application.

Figures 2(a) and 2(b) represent the entire life-cycle of the application and the ArmDroid options menu, respectively.

2.2 Application settings and preferences

ArmDroid needs to connect to a remote server node that, physically or virtually, operates the KUKA youBot. For that, ArmDroid needs to know the URL and the listen port of the server node it is connecting to. These data are specified by the user by accessing the entries `Gazebo address` and `Gazebo port` in the preference menu, respectively.

The `TeachPendantActivity` requires that the user provides joint angles to move the robot arm. These values can be expressed in different units of measure, e.g., degrees or radians, according to the user's preferences. For that, ArmDroid provides the `ListPreference Angle units` in the preference menu.

Finally, the user may want to lock the app screen to the current orientation. This is done by ticking the checkbox named `Lock orientation` in the preference menu. Note that this option only affects the `TeachPendantActivity`, since the

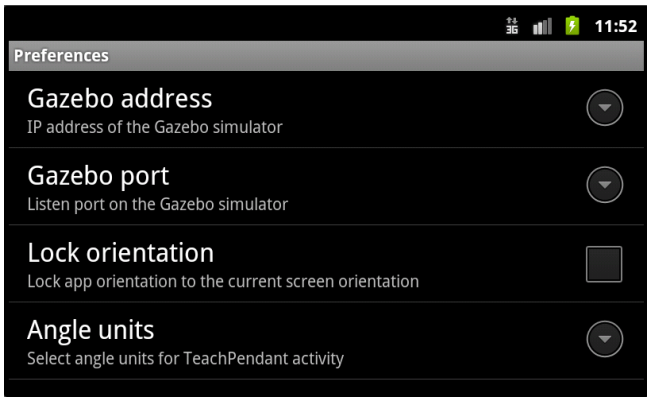


Figure 3: The ArmDroid preference menu in landscape mode.

screen orientation of the `MobilePlatformDriveActivity` is always locked to landscape mode by itself.

Figure 3 shows the preference menu in landscape mode.

2.3 Robot arm teleoperation

There are three different ways to specify a joint set configuration to be sent to the remote server node that operates the robot. These are implemented by three different regions within the user interface of `TeachPendantActivity`. With reference to figure 4, these regions are enclosed by green, orange and yellow lines. Note that, only values within the range $[-180, +180]$ degrees (or $[-\pi, +\pi]$ radians) can be assigned to joint axes; values that fall outside this range are set equal to the maximum and minimum of the range.

The green-line region encloses a set of push buttons that *atomically* increment (plus) or decrement (minus) the joint value of the corresponding axis. Joint values are incremented/decremented by a fixed amount: 5 degrees, when the `Angle units` preference is set to degrees, and 0.1 radians, otherwise.

The orange-line region contains a set of edit texts, each of them allowing the user to directly specify a value for the corresponding joint axis.

Finally, the yellow-line region frames a set of push buttons that allows the user to manage a local database. Previously saved joint set configurations can be loaded at any time during the life-time of the activity, or deleted when are no longer considered of interest.

2.4 Mobile base driving

The "Wii-mote"-like mode, realized by the `MobilePlatformDriveActivity`, assumes that the user uses its Android-powered device as a game controller. The screen orientation, locked to landscape, as in figure 5(a), enforces the idea that the user has to hold the device like a small steering wheel, which can be rotated around three different axes: roll, pitch and yaw[4]. Longitudinal, transversal and rotational (angular) motions of the mobile robot base are automatically inferred

[4] `http://en.wikipedia.org/wiki/Aircraft_principal_axes`

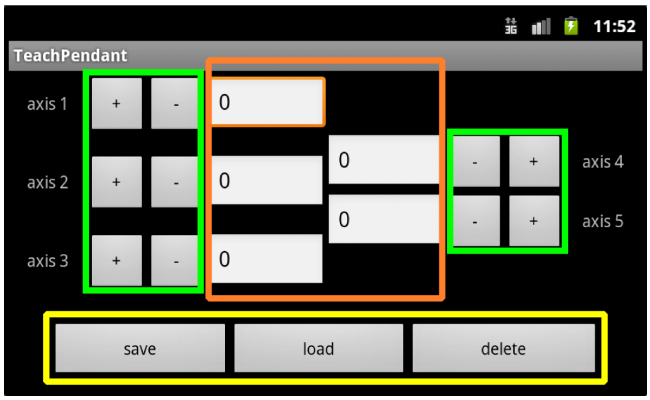
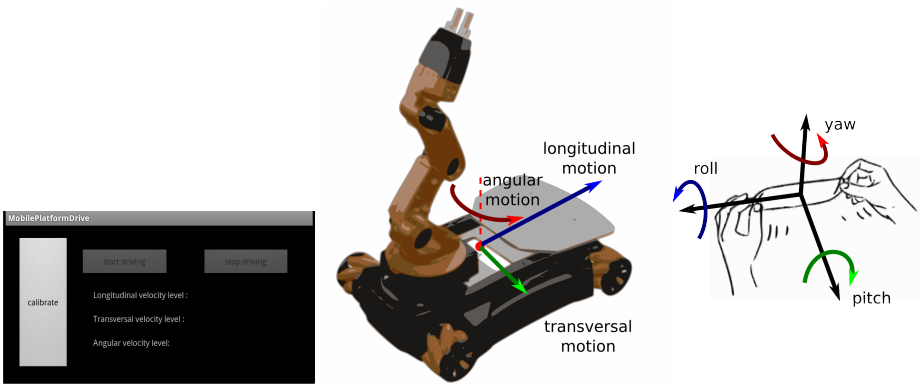


Figure 4: User interface of `TeachPendantActivity` with green-, orange- and yellow-line regions.



(a) (b)

Figure 5: (a): User interface of the `MobilePlatformDriveActivity` in `IDLE` functioning mode. (b): Map of changes in the device's orientation into robot's movements.

by ArmDroid, on the basis of the sensory information provided by the device when the user rotates it around that axes. Figure 5(b) illustrates how changes in the device's orientation are mapped into robot's movements.

The `MobilePlatformDriveActivity` has two functioning modes: `IDLE` and `DRIVING`. When in its initial functioning mode, the `IDLE` mode, the activity does not send any motion data to the remote server node, and it only displays the device's orientation on the screen. When the `calibrate` button is pressed, the activity saves the current device's orientation (instantly) and sets it as the initial (zero) orientation, still remaining in `IDLE` mode but now allowing the `start driving` button to be pressed. Each change of the device's orientation is now computed with respect to this "zero configuration".

When the user presses the `start driving` button, the activity switches to the `DRIVING` mode. Sensory data from accelerometer and magnetometer are periodically sampled and filtered, in order to transform them to "velocity

levels" for longitudinal, transversal and angular motions. Velocity levels are pure integer values ranging from +2 to −2, where 2 indicates "fast", 1 stands for "normal" and 0 is for "no" motion. The sign indicates the way of motion, i.e., forward/backward for longitudinal, left/right for transversal, left/right on spot for rotational motions. Of course, motions can be combined in order to enable the KUKA youBot to move along an arc (left/right arc) as well.

The user can stop the driving procedure and return back to the `IDLE` mode, by pressing the `stop driving` button at any time.

3 ArmDroid internals

After having introduced ArmDroid (section 1) and described how to get started with it (section 2), this section takes a look at its internal implementation and design choices.

3.1 Activities

The `TeachPendantActivity` provides the user with edit texts and push buttons to specify the robot arm joint configuration to be sent to the remote server node. Each edit text allows the user to directly specify a value for the corresponding joint axis. For this purpose, edit texts register `setOnFocusChangeListener`s and implement the abstract public method `onFocusChange`, called when the focus state of the view changes. "Plus" ("minus") push buttons, that atomically increment (decrement) the joint values, all call the same callback when clicked, namely `teachPendantButtonPlusAxis` (`teachPendantButtonMinusAxis`). There, the joint value to be modified is selected on the basis of the button ID. Callbacks are assigned to push buttons in the XML layout, using the `android:onClick` attribute.

The `MobilePlatformDriveActivity` relies on the sensory information from accelerometer and magnetometer to compute the device's orientation. For this purpose, the activity uses an instance of `SensorManager`, that gives access to the device's sensors, and implements `SensorEventListener`.

```
/* MobilePlatformDriveActivity */
...

protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(
        this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
                SensorManager.SENSOR_DELAY_GAME );
    mSensorManager.registerListener(
        this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
                SensorManager.SENSOR_DELAY_GAME );
}

protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
}

public void onSensorChanged(SensorEvent evt) {

    ...

    if (type == Sensor.TYPE_MAGNETIC_FIELD) {
```

```
        geomag[0]=evt.values[0];
        geomag[1]=evt.values[1];
        geomag[2]=evt.values[2];
    } else if (type == Sensor.TYPE_ACCELEROMETER) {
        gravity[0]=evt.values[0];
        gravity[1]=evt.values[1];
        gravity[2]=evt.values[2];
    }

    if ((type==Sensor.TYPE_MAGNETIC_FIELD) ||
            (type==Sensor.TYPE_ACCELEROMETER)) {
        SensorManager.getRotationMatrix(rotationMatrix, null, gravity, geomag
          );
        // compute the sought device's orientation
        SensorManager.getOrientation(rotationMatrix, rpy);
        ...
    }

    ...

}
```

Event listeners for both the sensors are registered in `onResume`, and are unregistered in `onPause` in order to avoid wasting battery power when the activity does not run. The core functionality provided by the `MobilePlatformDriveActivity` is coded in `onSensorChanged`, where the `getRotationMatrix` method is used, which takes the values from the accelerometer and magnetometer and returns a matrix that is finally processed by `getOrientation` to determine the sought device's orientation.

3.2 Application

ArmDroid's activities need to share settings and data structures during their lifetime. In order to avoid code duplication and to achieve a modular and extensible design, ArmDroid *extends* the `Application` base class provided by the Android framework and move all the common functionality to `ArmDroidApplication`.

```
/* ArmDroidApplication.java */

public class ArmDroidApplication extends Application
        implements OnSharedPreferenceChangeListener {

    // app-level parameters
    SharedPreferences preferences;
    InetAddress simServAddr; // server URL
    Integer simServPort;     // listen port

    ...

    public void onSharedPreferenceChanged
        (SharedPreferences sharedPreferences, String key) {
        ...
    }

}
```

Common settings include the remote server configuration parameters (URL and listen port). An example of data structure that is common to the two activities is the `SharedPreferences` object. Another example is described in the next subsection.

3.3 Periodic actuation jobs

In both the activies, the system configuration is periodically sampled, eventually filtered, and finally sent to the remote server node via UDP. The system configuration for the `TeachPendantActivity` is the robot arm joint set, whereas the device's orientation and the generated twist[5] of motion are the system configuration for the `MobilePlatformDriveActivity`. But this is not the only difference in how the UDP packets are generated by the two activities. In fact, packets must also include an identification string to specify to the server whether the received motion specification is for the robot arm or the mobile base. This suggests the use of an *abstract* class defining the behaviour of a "base" actuation job, whose *concrete* implementations, one per activity, differ in the way they construct the message for the server. With reference to figure 6, the following code is implemented in ArmDroid.

```
/* ArmDroidApplication.java */
...
public abstract class BaseJob extends TimerTask {

    protected String msg;
    public Handler handler = new Handler();

    public abstract void buildMessage();

    public void run() {

        handler.post(new Runnable() {
            public void run() {
                buildMessage(); // construct msg
                DatagramSocket simServSock = new DatagramSocket();
                ...
                DatagramPacket simServPack = new DatagramPacket(...); // msg
                simServSock.send(simServPack);
            }
        });

    }

}

/* TeachPendantActivity.java */
...
public class ArmActuationJob extends BaseJob {

    ...

    public void buildMessage() {
        msg = Arrays.toString(armDroidApp.tpStruct.getStatus()) +
                ", " + armDroidApp.tpStruct.angleUnits + ", arm";
    }

}

/* MobilePlatformDriveActivity.java */
...
public class MobilePlatformActuationJob extends BaseJob {

    ...

    public void buildMessage() {
        if (activityMode == ActivityMode.DRIVING)
            msg = Arrays.toString(armDroidApp.mpDriveStruct.generateTwist())
                    + ", mp";
    }

}
```

[5] http://en.wikipedia.org/wiki/Screw_theory#Twist

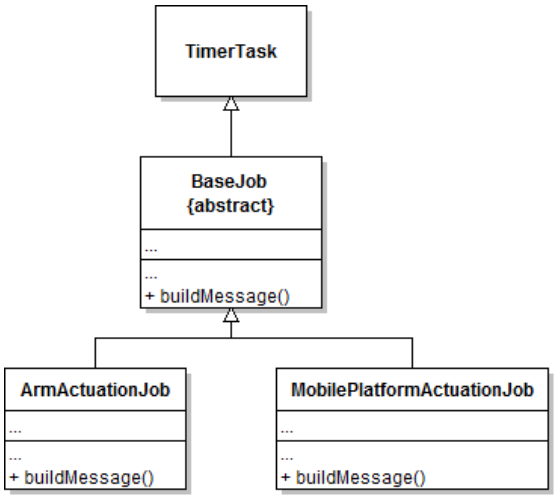


Figure 6: Simplified UML representation of the data structure implementing actuation jobs in ArmDroid.

Actuation jobs *extend* `TimerTask`, a class provided by the Android framework to represent a task to be run at a specified time, once or repeatedly. To preserve system resources, ArmDroid combines the actuation job timer tasks with a `CountDownTimer` that keeps track of how much time has passed since the last user/activity interaction and that cancels the timer task if no interaction has occurred within the last 5 seconds. Cancelled timer tasks are re-created and restarted by `fireUpAnActuationJob`, each time a method that changes the system configuration is executed.

3.4 Database

The `TeachPendantActivity` provides functionalities that allow users to save, load and delete joint configurations in/from a local database.

Until now, important Android building blocks such as services and broadcast receivers have not been used for developing the app. In fact, Android services are used for processes that should run *independently* of activities, which may come and go. But since ArmDroid does not comply to this logic in any of its functionalities, it has been considered a good practice to not use services (and broadcast receivers) in this context. On the other hand, to use as much Android building blocks as possible was a goal of the project, thus database functionalities have been implemented by using services and broadcast receiver, even if this may appear a bit overused in this case.

Each service creates an `AsyncTask` to perform all the database operations (create, read and delete) in a thread different from the UI thread. These operations are all handled by the `TeachPendantDataBaseHandler` class that extends `SQLiteOpenHelper`. It operates on `TeachPendantJointConfiguration` objects that encapsulate the information about the robot arm joint set configurations and *implement* the class `Parcelable`, so as to allow data serialization and favour a clever design.

The following code shows (portions of) the joint set configuration importer

as implemented in ArmDroid.

```
/* TeachPendantActivity.java */
...

protected void onResume() {

    super.onResume();

    ...

    // register receiver for db queries
    IntentFilter serviceActiveFilter = new IntentFilter(
      QUERY_COMPLETED_BROADCAST);
    this.serviceReceiver = new BroadcastReceiver() {
        public void onReceive( final Context context, Intent intent ) {
                // receive a configuration object (Parcelable)
                TeachPendantJointConfiguration jc = (
                  TeachPendantJointConfiguration)
                    intent.getParcelableExtra("jointSetConfiguration");
                armDroidApp.tpStruct.setStatus(jc.getStatus());
                fireUpAnActuationJob();
                writeEditTexts(jc.getStatus());
        }
    };
    this.registerReceiver( this.serviceReceiver, serviceActiveFilter );

}

protected void onPause() {

    super.onPause();

    ...

    // unregister receiver
    if (this.serviceReceiver != null) {
        unregisterReceiver(this.serviceReceiver);
        this.serviceReceiver = null;
    }

}

public void importJointSet(View v) {

    AlertDialog.Builder builder = new AlertDialog.Builder(this);
        ...
        .setPositiveButton("Import", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                String jointSetName = jointSetEditText.getText().toString();
                Intent buttonIntent =
                    new Intent(TeachPendantActivity.this,
                        TeachPendantDataBaseHandlerQueryService.class);
                buttonIntent.putExtra("jointSetName", jointSetName);
                TeachPendantActivity.this.startService(buttonIntent); //
                  query
            }
        }).show();

}

/* TeachPendantDataBaseHandlerQueryService */
...

public class TeachPendantDataBaseHandlerQueryService extends Service {

    TeachPendantDataBaseHandler dbHandler;

    ...

    public void onStart(Intent intent, int startId) {
        // get the name of desired joint set
        String jointSetName = intent.getStringExtra("jointSetName");
        // start the actual query task (AsyncTask)
```

```
        this.new ReaderTask(jointSetName).execute();
    }

    private class ReaderTask
        extends AsyncTask<Void, Void, TeachPendantJointConfiguration> {

        ...

        protected TeachPendantJointConfiguration doInBackground(Void... arg0)
           {
            // performs the query on dbHandler
            //      SQLiteDatabase db = this.getReadableDatabase();
            //      cursor = db.query(...)
            //      <save the TeachPendantJointConfiguration>
            //      cursor.close();
            //      db.close();
            //      <return the TeachPendantJointConfiguration>
            return dbHandler.getJointConfiguration(jointSetName);
        }

        protected void onPostExecute(TeachPendantJointConfiguration jc) {
            if (jc == null)
                ... // not found
            // call the private method that sends broadcast:
            //      Intent intent = new Intent();
            //      intent.setAction(QUERY_COMPLETED_BROADCAST);
            //      intent.putExtra("jointSetConfiguration", jointConfig);
            //      this.sendBroadcast(intent);
            sendQueryServiceCompletedBroadcast(jc);
        }

    }

}
```

4 Conclusions

This document described ArmDroid, an Android application that allows to teleoperate the KUKA youBot mobile manipulation platform. Developed as final project for the course "Android Framework" taught at Scuola Superiore Sant'Anna, ArmDroid is designed so as to provide a simple interface and concrete (even if basic) teleoperation functionalities.

In spite of its simplicity, the underlying idea is interesting considering the increasing adoption of Android devices worldwide.

As a future work, a specific activity should be implemented that allows the robot gripper (tool or hand) to be teleoperated, at the same way that the robot arm and the mobile base are today. In this way, the robot will be able to interact with the surrounding environment and perform complex tasks.

Furthermore, it should also integrate some form of visual feedback of the robot point-of-view, e.g., from sensors such as Microsoft Kinect. In this way, it would result a great tool used by human operators for helping autonomous robots accomplish demanding tasks in the real-world.

References

[1] KUKA youBot Store. Available: http://youbot-store.com/, 2011.

[2] The Gazebo Simulator. Available: http://gazebosim.org/, 2011.

[3] Android Framework, Scuola Superiore Sant'Anna. Available: http://retis.sssup.it/~panizzo/android/, 2012.

[4] ROS- Robot Operating System. Available: http://ros.org/, 2012.

[5] Locomotec. KUKA youBot User Manual. Available: http://youbot-store.com/downloads/KUKA-youBot_UserManual_v0.83.pdf. Technical report, November 2011.