Using the Android Platform to control Robots

Stephan Göbel, Ruben Jubeh, Simon-Lennert Raesch and Albert Zündorf
Software Engineering Research Group
Kassel University
Wilhelmshöher Allee 73
34121 Kassel, Germany
Email: [sgoe | jubeh | lrae | zuendorf]@cs.uni-kassel.de

Abstract—The Android Mobile Phone Platform by Google becomes more and more popular among software developers, because of its powerful capabilities and open architecture. As its based on the java programming language, its ideal lecture content of specialized computer science courses or applicable to student projects. We think it is a great platform for a robotic system control, as it provides plenty of resources and already integrates a lot of sensors. The Java language makes the system very attractive to apply state-of-the-art software engineering techniques, which is our main research topic. The unsolved issue is to make the android device interoperate with the remaining parts of the robot: actuators, specialized sensors and maybe co-processors. In this paper we discuss various connection methods and present a first approach to connect Android with the LEGO Mindstorms NXT robotics system, which we successfully used in our robotics/software engineering courses so far.

I. INTRODUCTION

Android devices are powerful mobile computers with permanent internet connectivity and a rich variety of built-in sensors. More properties make the Android system very applicable for university use: Android uses the Java programming language, which our students are familiar with. Getting started with the Android API is easy; the API is open, i.e. developers can access almost every low-level function and are not sandboxed. In addition, the Android API allows easy access to the hardware components. Interesting for robotics use are the numerous communication interfaces like WiFi, Bluetooth and GSM/UMTS, USB, and the integrated sensors, that is: accelerometer, gyroscope, compass and GPS. Because its a mass product, devices are available for already around 100$, which is much cheaper than any other ARM-based processing unit (e.g. Beagle Board). But the Android platform currently lacks the ability to physically extend it to control more sensors and actuators. This is actually a precondition if we want to use an android device as robotic processing unit, and section VI-A will discuss various options to overcome this restriction.

As we are software engineers, the main focus of our robotic related courses lies in software aspects like model driven software development, code generation, test based development, and strict object orientation. To make algorithms, data structures and software behavior more concrete, we started to create a bridge to real world objects by the use of robotics. Because it’s easy to build robots with, we initially used the LEGO Mindstorms RCX and later NXT for our projects and courses. Mindstorms NXT allows to control up to three servo motors and provides a set of useful sensors, which is sufficient for building simple robots like path finders, forklifts etc. From our point of view, another advantage of the NXT system is the availability of a Java Virtual Machine, called leJOS. However the leJOS Java (no reflection), the CPU power and the RAM and ROM space (64kb each) provided by the NXT are quite restricted. Due to our experiences, the capabilities of the NXT do not suffice to run complex Java programs with complex runtime data models that want to use for smart system behavior. The LEGO Mindstorms NXT and leJOS will be further discussed in section III.

To overcome the restrictions of LEGO Mindstorms NXT while still using their sensor and actuator control capabilities, we use a two layer approach. The lower layer uses NXT controlled sensors and actuators and the upper layer provides the more complex behavior exploiting the capabilities of an Android device. The two layers are e.g. connected using Bluetooth. The NXT provides connectivity via Bluetooth or USB. As Android provides USB and Bluetooth as well, we use these communication methods to combine Android and the LEGO Mindstorms NXT. This is the key idea of this work, which discusses several connection methods and presents one technical solution to interface the two systems. Benefits are obvious: Android brings in much more processing power, plenty of RAM, integrated sensors, various wireless connectivity and can be easily extended with gigabytes of flash memory. It simplifies programming a lot, for example, processing the live camera image is much easier to implement than within the NXT system. Another argument to use an Android device in combination with the NXT is that sensor integration is much easier. Android sensors are build in, they are already power-optimized which might be beneficial for certain robot types (e.g. in combination with a quadcopter), the whole device is autonomous because of the integrated battery.

In future versions of our robots, we plan to replace the NXT completely for the lower layer by a microcontroller platform, e.g. the popular Arduino board 1. For the tasks left to the NXT, it is quite expensive, which makes it less attractive for low-cost robot solutions. Thus, we plan to get rid of the Lego Mindstorm NXT brick and just use the NXT motors and LEGO technic construction features for robots, which forms an attractive, low-cost but powerful and extensible basis for

1http://www.arduino.cc/
robots. Arduino boards provide open source software solutions to control LEGO sensors and motors. In addition, Arduino boards provide connectivity for many other cheap sensors and actuators. This would lower the cost of a robot even more: a sufficient android device plus the Arduino board costs less than a LEGO Mindstorms NXT brick (but already has sensors integrated and provides much more resources).

Currently, we have a number of LEGO Mindstorm NXT bricks still in our lab. Thus we still use them for basic sensor and actuator control. In this paper we present a technical solution to make Android and Lego Mindstorms NXT interoperable, in form of a software library. This library is called LPCCA (LeJOS-PcComms-Android) and is responsible for the connection between the two systems. It provides a powerful, object-oriented API on the Android side, based on the LeJOS API resp. the FujabaNxtLib, see section V. This library is designed to be usable for an Arduino based basic layer, too.

II. RELATED WORK

While we were working on this library, similar approaches showed up. Most of them don’t focus on the specific connection between Android and NXT, but rather on Android and microcontrollers in general, or to bring physical, wire bound microcontroller-enabled interfaces to Android. As a microcontroller, most projects refer the popular Arduino platform as de-facto standard for a microcontroller-based low-cost platform. It consists of an Atmel AVR CPU with a USB programming interface and a certain layout that allows easy extension of the base circuit board by adding standardized so-called shields (stackable circuit boards) on it. Arduino is programmed in a simplified C/C++ dialect and comes with its own IDE (called Arduino as well). There are different kinds of board designs available, e.g. the Arduino Mega Board comes with a bigger CPU and more I/O ports than the standard Arduino.

A. MicroBridge

The first related approach is a software project called MicroBridge 2 which builds upon the following hardware: an Arduino microcontroller board and a USB Host Shield. The USB Host is required because almost every Android device is a USB slave. The MicroBridge software emulates the host side of the ADB (Android Debug Bridge) protocol. This protocol can be used to transfer arbitrary data between the android device and the host, in this case, the AVR CPU. This project deals only with the low-level connection via USB and ADB and doesn’t add any higher-level communication upon it. The developer now has to implement some meaningful communication between the AVR and the Android device to read sensors, control actuators etc by using a virtual TCP connection between host and device. The major advantage of this approach is that it works with almost every android device, even version 1.x. The device itself remains unchanged for successful operation. Just the “ADB debugging” feature has to be activated.

B. IOIO

The IOIO Board 3 is a direct extension of the Android device and comes as hardware circuit board. The project also provides powerful android software and API. Like the MicroBridge approach, it connects to the device as USB host, but it does not feature a user programmable microcontroller CPU. The onboard PIC CPU has a fixed firmware. The idea is to control the boards I/O ports directly via the Android host. Therefore, a powerful Java API is provided. With that API, its easy to directly access the boards general I/O pins or any dedicated communication pins for e.g. SPI, I2C and Serial communication. Internally, it also stacks upon the ADB protocol and a virtual TCP connection. To connect this to NXT components, you would still have to implement for example direct NXT sensor readings as Android host software. But we think the IOIO approach is the most general one to connect any electrically interfaced hardware, actuator or sensor, to Android.

C. Google ADK

Google recently introduced a very similar approach which looks like a combination of MicroBridge and IOIO: The Android Open Accessory Development Kit (ADK) 4 consists on the hardware side of an Arduino Mega with the USB Host Shield integrated. Google also provides an extension shield which adds buttons, a joystick, relays etc. to the base board. ADK also contains a device API, but that one comes only on the newest devices (Android version 2.3.4+).

D. Cellbots

The cellbots.com 5 group, which arose from the 20%-Google-employee-free-time, was the first one to show some interoperability between Android and robot hardware. First experiments used the debug serial port of developer phones to connect to a microcontroller board via serial connection. Meanwhile, there’s a wide range of software and hardware development projects available from cellbots. Because the first debug-serial-connection approach was limited to a few phone types, an intermediate workaround idea was to use the headset (and microphone) connector to talk to an external microcontroller. Using a modulated signal, the TRRSTAN kit 6 can control up to four servo motors. Meanwhile, cellbots supports many robot platforms, including Lego Mindstorms NXT, iRobot Create und Roomba, TRRSTAN etc. There is a user-friendly Android Market Application, which uses Bluetooth connections to connect to various robots. It mimics a remote control and can control a NXT without any programming directly by user interactions. A more complex usage scenario requires two android devices, one as user interface, the other one as robot controller, preferably directly attached to the robot. For software developers, Java and Python libraries are available. There’s also an App Inventor plugin available which enables cellbots robotic features within the web-based App

References:

Inventor Tool, that is used to graphically program or design Android apps with little knowledge of the internal structures.

E. Amarino

The Amarino Toolkit \(^7\) is a library to connect Android devices to Arduino boards. It is limited to Bluetooth connections and requires either the Arduino variant of Arduino, called ArduinoBT, or a Bluetooth extension shield. With the library, Android applications can send data to the Arduino, e.g. their sensor values. As the library doesn’t provide any return channel, it doesn’t seem applicable for our purposes. Furthermore, being bound to just low-bandwidth, high-latency Bluetooth connections might not fit all application examples.

F. leJOS 0.9/Android

Since version 0.9 the leJOS library supports Android directly. This is done by adapting to the Android Bluetooth API, similar to what we did it. But upon that, there’s just a single class encapsulating LCP command creation for remote controlling a NXT, so this approach lacks a powerful API, which we think is necessary for more complex robot control applications.

The three USB-related developments, MicroBridge, IOIO and specially Google’s own ADK approach, are another reason why we concentrated on the Bluetooth interoperability between Android and NXT first. All three are currently under heavy development or have just been announced, so we’re waiting until the situation stabilizes.

III. LEGO MINDSTORMS NXT

The LEGO Mindstorms NXT system has become very popular in schools and universities. It is cheap (compared to more professional robot platforms), it’s very flexible because it comes with a wide range of mechanical elements of which you can build a variety of robots, and it comes with an easy-to-understand graphical programming software. LEGO provides a basic set of sensors (ultrasonic range sensor, light sensor, buttons). In addition, there are different kinds of compatible sensors available from 3rd-party manufacturers. The LEGO actuators are continuous servo motors, which are flexible and powerful enough to power a vehicle but also accurate enough to move a robotic arm within the accuracy of a degree. The system features a central component, the LEGO Mindstorms NXT brick, as processing unit. It contains a 32-Bit-ARM7-CPU, a LC-Display, some buttons for user interaction and a rechargeable battery. Via RJ11 wires, up to three motors and four sensors may be directly connected. The brick can communicate via USB or Bluetooth e.g. with a host PC or with other NXTs. Programs can be uploaded via both connection methods. After being programmed, the brick is autonomous. LEGO provides a programming environment called NXT-G. It features a graphical programming language which is sufficient for very simple robotic programs. Because the brick specification was opened by LEGO, there’s a wide range of alternative firmwares, development environments and libraries for various languages (C, Assembler, Java, Matlab...). Most of them are open-source. We use the Java firmware implementation, called leJOS \(^8\).

IV. LEJOS FOR NXT (NXJ)

With the development of leJOS (Java for LEGO Mindstorms, available as RCX VM for older systems as well for the NXT), it became not only possible to remote control NXT from computers running java but also to deploy native Java programs to the NXT brick and to run them locally. In order for this to work, leJOS consists of several parts. The most important part is the alternative firmware for the NXT. This leJOS firmware is loaded into the bricks flash memory and permanently replaces the original LEGO firmware. The leJOS firmware has been completely rewritten in C and ARM assembler and consists of drivers for the hardware and the Java Virtual Machine (JVM) to run native Java bytecode. The startup menu is already a native Java program which is capable of loading other Java programs from the systems flash storage into the JVM. The JVM is running one program at a time, which means it is replacing the old one when a new one is loaded. The local programs are stored in the internal storage, with read-only parts of the code being left in the flash storage and read-write parts of the program copied to a heap in the RAM which is watched by a garbage collector. The leJOS API provides the means to write Java programs running on the brick that interact with the sensors and motors directly. The leJOS PC API, while organized very similar, is used for running Java programs on a computer and remotely control the NXT via Bluetooth or USB connection, utilizing the LEGO Communications Protocol (LCP) over Java stream connections. The connection to the brick is handled by classes in the lejos.pc.comm package, which is only part of the leJOS PC API and not included in the plain leJOS API.

V. FUJABA NXTLIB

Fujaba \(^9\) is a graphical UML CASE tool based on Story Driven Modeling (SDM) \([1]\) methodology. This is a software development approach where software functionality is specified in so called Story Diagrams, among standard UML diagram types like class diagrams. An adaptable code generator generates Java source code out of the graphical diagrams. The methodology supported by Fujaba focuses on strictly object oriented, example- and test driven development. In \([2]\) was shown, that this software development approach gets more intuitive when brought into the real world. A forklift robot was solving the Towers of Hanoi game. This example was mainly used in high schools for first programming education. In \([3]\) we

\(^7\)http://www.amarino-toolkit.net/

\(^8\)http://lejos.sourceforge.net/

\(^9\)http://www.fujaba.de/
created an object oriented library called *FujabaNxtLib* upon the leJOS framework, which was modeled within Fujaba Story Diagrams, to control LEGO Mindstorms NXT robots. It adds an abstraction layer upon the leJOS PC API. The generated application code runs on a standard desktop PC and connects to the NXT via bluetooth, so we’re remote controlling the NXT. This way, we can create more complex programs as we are not bound to the NXT hardware limits (64kb RAM/ROM). Debugging is possible, we can see and graphically represent the software internals (using a graphical heap visualizer, called eDOBS [4]) and use interactive, stepwise development of the control algorithms, see [5]. This plays benefits especially for educational purposes, because graphical representations are easier to understand and have a more direct reference to real world objects, in this case, our robots and their components. Fujaba in combination with NXT robots for educational purposes has been also used by [6] resp. [7]. The FujabaNxtLib added the additional abstraction layer for another reason: different adapters, either to leJOS or to virtual sensors and motors, can be plugged in without changing the robot control code at all. This way, we can plug in a simulation layer and run our code independent of an actual robot for testing and simulation purposes.

Figure 2 shows an excerpt of the FujabaNxtLib API. FNXT represents the NXT brick. Connected to it by one association per port are the motors and sensors. Every sensors derives from the FSensor superclass and implements the observer pattern (not shown in the diagram). The FMotor class controls a single motor, which has an integrated rotation encoder. The FNavigator class, developed for driving robots with two motors, abstracts the motor control and adds higher-level functions for rotating, turning and driving the robot by delegating to the two driving motors. The navigator can be configured for a certain wheel size, track width and gear reduction, because of that it can move the robot predictably in absolute coordinates. The navigator is the central class for driving control, so we will concentrate on that class in the following sections and don’t show the sensor event handling etc.

### VI. LPCCA Bluetooth library

In order to establish useful connections between Android and NXT, a library had to be developed. Its purpose is to manage the connection efficiently and provide a developer-friendly way of using the connection, i.e. have some useful API, which led to wrapping the leJOS PC API so it could be used from the Android device. The library is designed as a tool for realizing new application ideas without having to worry about the connectivity and using an already established API. An example of how this library can be used for new applications is the *WebMoteRobot*, which will be presented in section VII. Other application examples, mainly remote controlled robots, are currently under development by this semesters students course.

**A. Connection considerations**

1) **Direct via USB**: On the first sight, USB seems to be the preferred connection method between Android and NXT. But the NXT is a USB slave device, and usually every Android device is a slave too. Therefore, USB connections can only be established by enforcing USB host mode (or USB OTG) or by adding a USB Host device in-between. The first solution is not supported by the vast majority of Android devices. The latter requires a micro-controller with USB host interface, which introduces new problems: first of all, some adapting bridge code has to be implemented. Then, it’s unclear how the intermediate host device gets powered. Furthermore, the different android firmwares might not support drivers for serial-over-USB-connections, so the device firmware needs to be modified. The main benefit of using USB would be that a wire-bound connection is quite fast (300kbit/s), has a low latency and is robust against radio interferences. Mainly because of the compatibility issues, we wait for new developments in this area and look for alternative solutions for now. Some Android devices offer direct serial ports on their connectors. This solution wasn’t pursued either, as we would limit our solution to certain device types.

2) **Indirect via microcontroller**: The second possibility is to use an indirect connection: a microcontroller board, e.g. Arduino, adapts between the NXT and the Android device. This idea was developed mainly because of ongoing talks about moving to a NXT-less robotics system, directly controlling sensors and motors with microcontrollers. It mainly arose as it is an even cheaper approach than the Lego Mindstorms system. However for easier and faster results it can also be used with the NXT still in place basically as a batterypack for the motors and easy connectibility. On the Android side we either have to use USB as the only wired connection method, or any wireless method, e.g. bluetooth. The connection between the NXT and the microcontroller board is usually established via the I2C protocol, attached to one of the NXT sensor ports. Having a microcontroller board connected to the Android device allows a wide range of flexible configurations, e.g. leave the NXT out and connect sensors and actuators directly to the microcontroller, or using it to multiplex between different NXT at once. This requires a lot of software implementation on all three system components. But this setup is too complex and more expensive than the direct connection between Android and NXT and should only be considered as a move towards NXT-less systems.

3) **Bluetooth**: The third possibility for interconnectivity is Bluetooth. Bluetooth based wireless connections are basically supported by all shipped Android devices today. All NXT firmwares support remote Bluetooth control via LCP (Lego Communications Protocol). This combination is the one big advantage over any other possible connection. leJOS already supports translating commands from its object oriented library to LCP transmission protocol commands and encapsulates for sending it over a SPP-Bluetooth connection. Just one class of the leJOS library, which is responsible for adapting to the
concrete systems bluetooth stack, had to be modified to get the LCP over Bluetooth-code working on Android. The main disadvantage of using a bluetooth connection is the high and jittering latency, limited throughput and sometimes stability issues. As most of our application scenarios are not really time critical, this is acceptable. On the other hand, using a wireless connection doesn’t require the Android device being close or wired to the NXT, so it can be instantly used as remote control. As we did only minimal modifications to the leJOS implementation, upgrading to new versions seems easy. Furthermore, we are not bound to NXT with the leJOS firmware: as LCP is used by the standard LEGO firmware and others, we can control even the standard LEGO NXT firmware with this approach.

B. Implementation

The library in its current implementation consists of a RemoteService called LPCCARemoteService. A remote service is an Android system wide software interface. Once started, it encapsulates and controls the connection to the NXT system-wide for all Android applications. This makes it possible to establish a connection in one Android application and actually use it in another one. The WebMoteRobot for example could be set up by someone using the Android device and an on-device application to establish the connection, which is then used by a web application running on the device also, but that is controlled remotely by another users browser. Such a setup is presented in section VII. An Activity (corresponds to an Android User Interface Screen) that can be started by the RemoteService provides a simple means of setting up the connection to the NXT, basically providing a list of all bluetooth-enabled NXT in discovery range. Once a decision for bluetooth pairing and connection was made, the Activity returns to the application that asked the RemoteService to start the Activity, enabling it to now control the NXT via the leJOS PC API. The use of this Connect-Activity is completely optional, each application can choose to establish the connection itself, asking the RemoteService for a list of available NXT, and telling it which one to connect to. To determine which NXT shall be used, it can either have its own implementation of a visual selection for the user, or just have some hard coded naming scheme, always connecting to the same NXT.

LPCCARemoteService extends the Android API class Service. It implements ILPCCARemoteService which is defined via an AIDL file. AIDL stands for Android Interface Definition Language and allows you to define the programming interface between separated Android processes, i.e. our service, and the client, newly developed applications, agree upon.

In order to provide a list of NXT only devices, all discovered devices are checked upon their mac address, filtering those that specify LEGO as the manufacturer, as the NXT is the only device with bluetooth build by LEGO. In order to notify applications using the library of newly discovered NXT the LPCCARemoteService sends its own broadcasts so interested applications can subscribe via BroadcastReceiver objects. This is useful if an application is started before the NXT is turned on and discoverable, because the NXT won’t be in the initially transferred list of available devices, but rather show up as a broadcast.

```java
package org.lpcca.service;
import org.lpcca.service.*;
interface ILPCCARemoteService {
```
List<String> getAvailableDevices();
void requestConnectionToNXT();
void establishBTPConnection(String deviceKey);
void requestDiscovery();
boolean isConnected();
Navigator getNavigator();
NXT getNXT();
Motor getMotor(char port);
}

The first four methods are used to initiate the bluetooth connection to the NXT. When being connected, the getter-methods can be used to retrieve a proxy object that represents the NXT, a motor or the navigator. In detail, the getNavigator():Navigator method returns a navigator instance, which is defined as AIDL interface as well, which follows in the next listing:

package org.lpcca.service;
interface Navigator {
  void forward();
  void rotateLeft();
  void rotateRight();
  void stop();
  void backward();
  void travel(double distance);
  void turn(double radius, int angle);
  void rotate(int degrees);
}

By declaring the complete FujabaNxtLib API layer via AIDL, it is accessible to Android applications, which can use the exposed proxy objects and don’t have to deal with the low level LCP protocol etc. Furthermore, this object oriented approach makes it easy to introduce non NXT components externally connected later on, or to multiplex and support multiple NXT connections at once.

An exemplary sequence for connecting an application using the LPCCA library and connecting to a NXT is shown in figure 3. All communication between the application, the remote service and the bluetooth subsystem is inter-process communication and therefore done via Android intents.

VII. WEBMOTE

As a proof of concept we have developed a small application that makes use of the provided functionality by the Android API. The application can be accessed via web and provides a videostream from the Android camera as well as an interface with buttons to control the robot. Using the camera and the wireless connection to access the web makes the Android device act as sensor and actuator. There are no NXT extensions that provide this functionality. The robot is to be build with the Android device mounted in place, preferably rotated around the cameras axis by a third motor. The first two motors are used for steering and movement. If setup correctly, and with the Android device connected to the internet it is now possible to visually explore the surroundings of the robot from any browser that has access to the internet. We called our application WebMoteRobot. It is implemented using the Google Web Toolkit (GWT)\(^\text{10}\). The web application is deployed on a Java web server called iJetty that is running on the Android device. The user can access the webapp via the IP address of the Android device and control the NXT with the provided buttons that are linked to the corresponding navigator methods of the LPCCA library.

Once a client connects to WebMoteRobot the server tries to establish a connection to the LPCCARemoteService, i.e. binding to the service. If the service was not started yet, it will be started now. Upon a successful connection it has access to

\(^{10}\)http://code.google.com/webtoolkit/
the NXT via the navigator and maps the buttons actions to the respective methods. To achieve this the functionality has once more to be wrapped as the clicks on the buttons are handled at client-side (the users browser), and connect to server-side (the Android device) via Remote Procedure Calls (RPC) that are defined as a GWT service. It not only wraps functionality but implements some further logic that makes using the web interface more simple, e.g. checking whether a connection has already been set up and in case it has not, calling the setup from the LPCCARemoteService. This makes using the web interface consistent, because it doesn’t affect the user whether a connection is still established by a previous session or if it is the first session. The whole interaction of the browser/client side, the web application on the device and the library service is shown in figure 6. With the Android device connected to the robot, changes in position and angle are visually fed back to the user via the web interfaces videostream. For easier navigation and to prevent unwanted movement, the buttons in the web interface move the robot by a predefined distance each click. This helps reducing latency induced errors that occur due to the lag between movement of the robot to actual reaction of the videostream in the web interface. It is not only the Bluetooth connection but mainly the wireless streaming of video that increase this lag. Another approach might have been reducing the constant moving speed, leading to problems if the internet connection is terminated, which is why we chose the predefined distances. It was possible to navigate through all of the rooms of the Software Engineering Research Group in Kassel University.

VIII. CONCLUSION

The LPCCA library introduced in this paper is fully functional, enabling developers to write Android applications that remote control NXT robots. Using the library, the Android may programatically determine complex control sequences for the NXT. The LPCCA library has been successfully introduced in our course Distributed Robotic Systems Modeling at Kassel University this term. It proved to provide simple access to standard LEGO robot sensors and actuators. Based on the LPCCA library it was easy to build complex Android applications that use object oriented data models in Java. The LPCCA library may however still be expanded beyond the actual state, making it more flexible and easier to configure.
Keep in mind that the library as of the writing of this paper is still a proof of concept and needs project specific adjustments to do more than basic controlling of the NXT. The following areas have been identified as being the most interesting ones to develop further:

- An Android controlled Segway (inverted pendulum principle). For real world results concerning the bluetooth induced latency, an Android controlled Segway would be very interesting to develop. The Android devices come with just the right sensors for a Segway, as it needs exact positioning data of the device, i.e. angle and orientation data provided by the accelerometer, gyroscope and compass sensors in the Android device. If the latency induced by Bluetooth prevents the Segway from staying upright, this would again motivate to go for USB based connections between Android and NXT or for an Arduino based solution.

- A configurator providing visual means of configuring the setup of the NXT, i.e. the usage of ports. This would increase the user experience as any changes to the wiring of the NXT can be adapted without changing the source code of the program, as long as the right sensors and motors are connected. The need for hard-coded port usage would be eliminated. This functionality should be provided by an Activity class in the library itself so it can be reused for all projects utilizing the library.

- USB support has yet to be implemented. Even though there is only a small subset of Android devices that are capable of this means of connection, it is an enhancement to the library, as some latency-related problems can be circumvented.

- Several NXT controlled by one device. The leJOS API is making heavy use of singletons in its current implementation, leading to a situation where only one NXT can be controlled at a time. However it is in general possible to use the bluetooth connection to control multiple NXT at the same time.

Altogether, the new LPCCA library provides a simple means for using Android devices to remote control LEGO Mindstorms NXT bricks and thus to bring complex computation capabilities to LEGO based robots. In addition, the Android devices add a lot of new sensors to LEGO robots like GPS, acceleration, and last but not least a camera and WIFI connectivity. And students love it.

REFERENCES


