AUTOMATYKA/ AUTOMATICS • 2013 • Vol. 17 • No. 2

http://dx.doi.org/10.7494/automat.2013.17.2.197

Krzysztof Kołek*

Application of Android OS as Real-time Control Platform**

1. Introduction

An android operating system, dedicated to mobile devices, notifies 250% annual growth in the number of active devices. It is estimated that the number of working devices exceeded 300 million, daily 850 000 new devices are activated, and the number of available, free of charge application is about 450 000 [1]. Among such a richness of hardware and software in a relatively small extent addresses the theme of the applicability of Android devices to control in real time.

This paper presents an application of an Android device to control a laboratory model of Anti-lock Breaking System (ABS). The architecture of Android OS is given. The features of the Android aimed to build real-time systems are discussed. ABS model with a mobile phone acting as a controller is shown. The aim of the control algorithm is to avoid locking the wheel. The controller is implemented in Java which is the basic development language in Android. The controller uses only standard system functions without the use of real-time extensions. The results of the experiments are shown. Conclusions include remarks on the applicability of devices running Android as a platform for building real-time control systems.

2. Android architecture

The success of the Android OS seems to stem from two facts: the nature of the system architecture and the software available on an open-source license.

Android's architecture is shown in Figure 1 [2]. Android is based on Linux kernel, currently at version 2.6. Linux functions related to security, memory and process management, network management and devices drivers of I/Os are directly ported to Android.

* AGH University of Science and Technology, Krakow, Poland
** Paper supported by AGH project nr 11.11.120.768

The *Library* layer is a collection of modules created in C/C++ in Linux exporting functions to Android. Libraries are optimized for portable battery-powered devices for efficient power consumption. The main modules of the layers are:

- libc – contains implementation of the standard C library functions;
- Media Framework – includes multimedia library applied to record and playback video, audio, and presentation of images in various formats;
- Surface Manager – manages the screen of the device;
- WebKit – implements the web browser engine;
- SGL – creates engine of 2D graphics;
- OpenGL – optimised 3D graphics library, optionally integrated with hardware accelerators;
- FreeType – supports raster and vector fonts;
- SQLite – contains lightweight SQL databases engine.

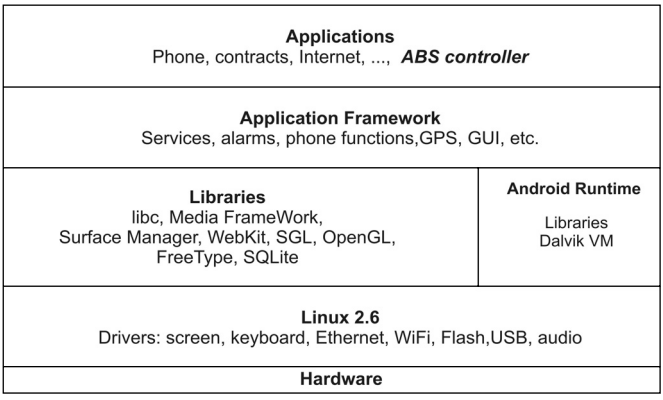

Fig. 1. Architecture of Android

The *Android Runtime* layer contains the software used in the Java programming language. Android uses optimized for mobile devices Java virtual machine called Dalvik. Each Android application runs as a separate process that runs its own copy of Dalvik machine. Dalwik machine calls Linux functions for the management of threads and memory.

The *Application Framework* layer provides for a wide range of services: access to the hardware functions, access to GPS data, running background services, set alarms, display messages and user application GUI services.

The *Applications* layer, depending on the type of device, includes applications to make phone calls, send SMS and MMS messaging, calendar management, maps browser, internet browser, etc. Applications are usually developed in Java. The controller of the laboratory ABS system also operates at the *Applications* layer. The controller is implemented fully in Java, using only the functionality of the *Android Runtime* and the *Application Framework* layers.

Although not used in the presented experiments, there should also be mentioned two ways of building C applications. The first way is derived from the fact that the Android is based on Linux. Running Linux applications requires the supervisor's permission, and is usually used only at the level of the system. The second way is to use a distributed system tool called the Native Development Kit (NDK). NDK allows the "wrapping" of C/C++ code to be able to be a direct called from Java programs.

The Android source codes are available at GNU license [3] and can be used for new equipment. Costs of adapting the system to a new device cover only the costs of the device drivers. This is an important reason, besides a rich set of features built into the system, of the growing popularity of the system.

3. Android functions applied for real-time

The implementation of real-time control tasks requires at least the following tasks:

- implementation of the communication with the hardware to perform measurements and set control signals to stimulate the object;
- execution of control tasks with a given sampling period, optionally raising the priority of the task;
- precise time measurement to estimate the punctuality of the control task;
- tuning the parameters of the OS to minimise the impact of factors such as system sleep or execution of the garbage collector (GC).

The nature of the Android devices affects the implementation method of the first task. Android devices are focused on wireless communications, most often without the possibility of connecting other peripherals. Selection of wireless communication seems all the more reasonable that the Java implementation of communication, for example using TCP/IP, is very simple [4].

The implementation of a constant period control algorithm can be performed using the timer mechanism. Activation of the cyclic call of a method associated with the timer is implemented in Java using the following statements:

```
Timer timer1 = new Timer();
timer1.scheduleAtFixedRate(new rtTask(), 0, Period_ms);
```

The statements create an object of class `rtTask` and activate cyclic execution of `run()` method of this object with a period of `Period_ms` milliseconds.

The priority of the timer task can be tuned by the `setPriority` method. There are only 10 levels of priority available for tasks executed by the virtual Dalvik machine. It seems to be a significant constraint for the development of complex control systems.

Checking the punctuality of the control task requires the precise time measurement. Android libraries contain the `System.nanoTime()` method that returns the time stamp.

The resolution of the time stamp is one nanosecond. Time stamps are dedicated to the measurement of time intervals, so the method can be used to determine the jitter of the control task.

Figure 2 shows the measurement of time intervals ΔT between successive entrances to the timer task. The timer period was set to 50 ms. This measurement allows specifying achievable task punctuality. The figure presents 1 million measurements made during a one-day phone usage. The average value of the ΔT was 49.999 ms, and the standard deviation was 1.510 ms. However, the figure also shows the ΔT values exceeding 250 ms. Large deviations from the set point were acquired during phone calls, Internet connections and activation of certain applications on the phone.

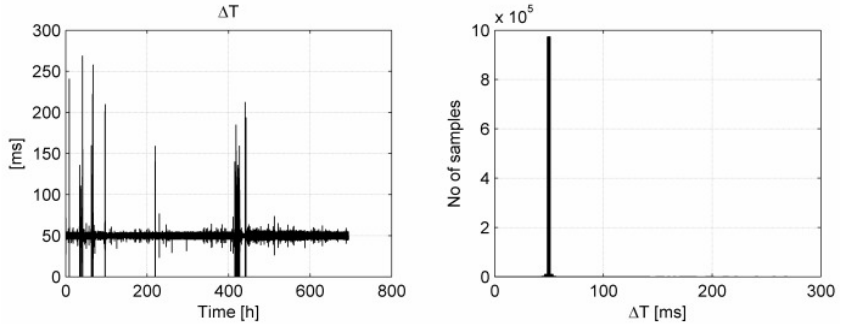

Fig. 2. Timer task jitter: time diagram and histogram

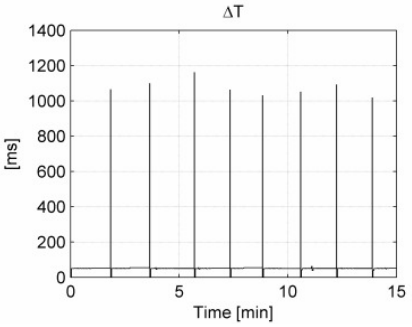

Fig. 3. Timer task jitter on the activation of suspend mode

The jitter rises significantly during the transaction to the suspend mode (Fig. 3). Due to the power saving policy the device periodically attempts to switch to the suspend mode. The measurements shown in Figure 2 were performed on the device maintained in an active state, without going to the suspend mode. Also during the experiments presented in this paper the suspend mode was disabled.

Figures 4 and 5 show the jitter histograms for the timer periods ΔT_{DES} equal to 10 ms, 50 ms, 100 ms and 500 ms for the minimum and maximum thread priorities. Table 1 shows

the quantitative parameters of the timer events. The table presents the mean value, standard deviation (RMS), the number of clock periods supported with errors smaller than 1 ms, 5 ms, 10 ms and 50 ms, and the maximum error of the timer for extreme priorities.

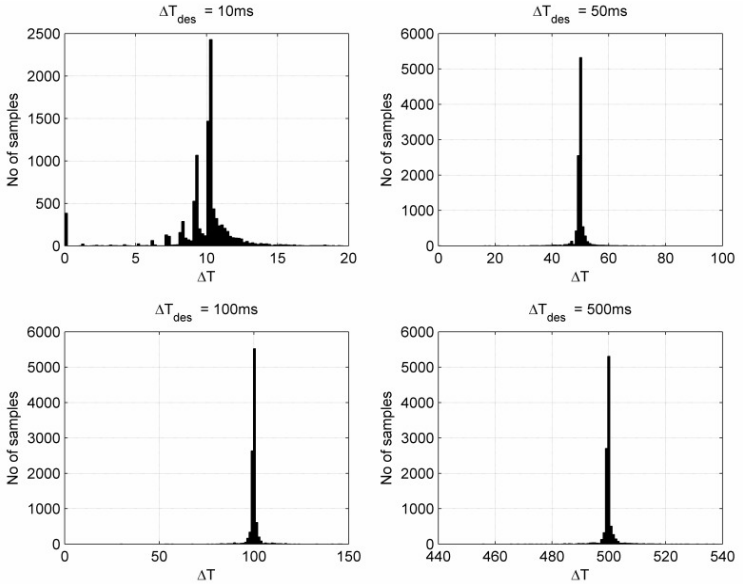

Fig. 4. Jitter histograms for maximum priority level (MAX_PRIORITY)

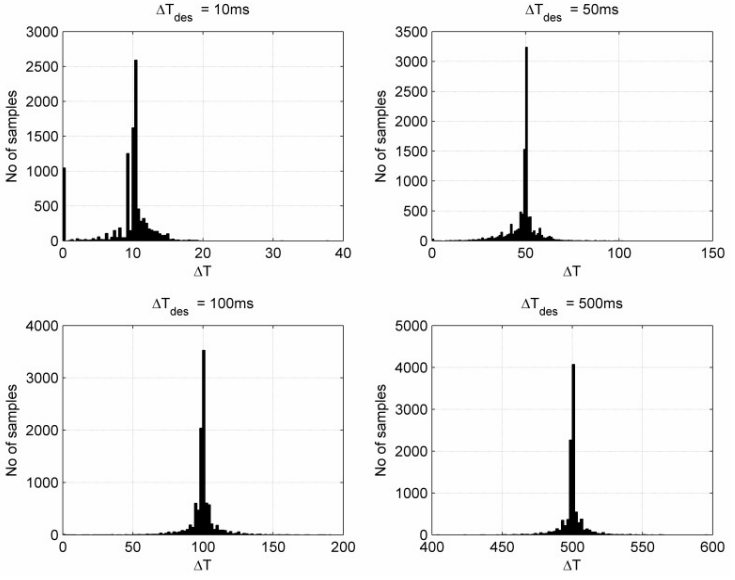

Fig. 5. Jitter histograms for minimum priority level (MIN_PRIORITY)

Table 1
Timer parameters

Maximum priority MAX_PRIORITY							
ΔT_{des} [ms]	Mean value [ms]	RMS [ms]	1 ms [%]	5 ms [%]	10 ms [%]	50 ms [%]	Max ΔT_{des} [ms]
10	9.71	2.50	69.66	94.01	100	100	9.99
50	49.91	2.49	82.03	96.20	98.56	100	49.99
100	100.03	2.93	82.04	96.20	98.01	99.98	70.74
500	499.99	2.66	83.85	95.97	98.36	100	44.63
Minimum priority MIN_PRIORITY							
10	9.22	3.77	60.92	86.18	99.98	100	27.90
50	48.53	8.47	47.36	71.43	85.06	99.99	52.19
100	99.69	10.79	49.02	75.69	85.89	99.24	99.99
500	499.99	8.71	51.66	75.41	87.33	99.60	96.69

In Java applications, there is a mechanism to remove unused objects called the Garbage Collector (GC). GC algorithm is complex and its work can significantly burden the system. The Android libraries contain the *System.gc()* method which suggests the system to run the GC. During tests the memory-heavy objects were created and periodically the *System.gc()* was activated. Such stress tests did not cause noticeable changes in the punctuality of the timer tasks.

4. Experiments with laboratory ABS model

As a test setup the laboratory model of Anti-Lock Breaking (ABS) is selected [5].

4.1. Laboratory ABS model

The laboratory ABS setup is shown in Figure 6. The model has a solid metal wheel to simulate the weight of the car and placed over it, pressed with a dumper, the wheel corresponding to the wheel of the car. Two wheel speeds are measured using encoders.

The wheels are accelerated by the driving DC motor to the linear speed of approximately 75 km/h, and then the hydraulic brake breaks the upper wheel. Doing so will decrease also the speed of the second wheel.

The aim of the experiments is to stop the wheels, minimizing braking distance while avoiding the lock of the upper wheel. The lock of the wheel is determined by the slip δ, calculated as follows:

$$\delta = \frac{\omega_{CAR} - \omega_{WHEEL}}{\omega_{CAR}} \tag{1}$$

where:

ω_{CAR} – velocity of the lower wheel (simulates car),
ω_{WHEEL} – velocity of the upper wheel (simulates wheel of the car).

Achieving the slip value of 1.0 means the lock of the upper wheel.

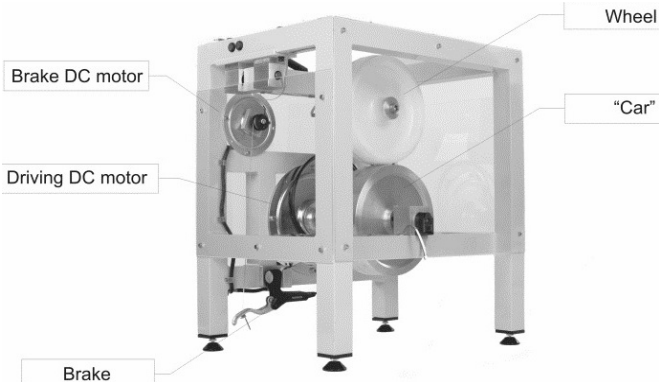

Fig. 6. Laboratory ABS model

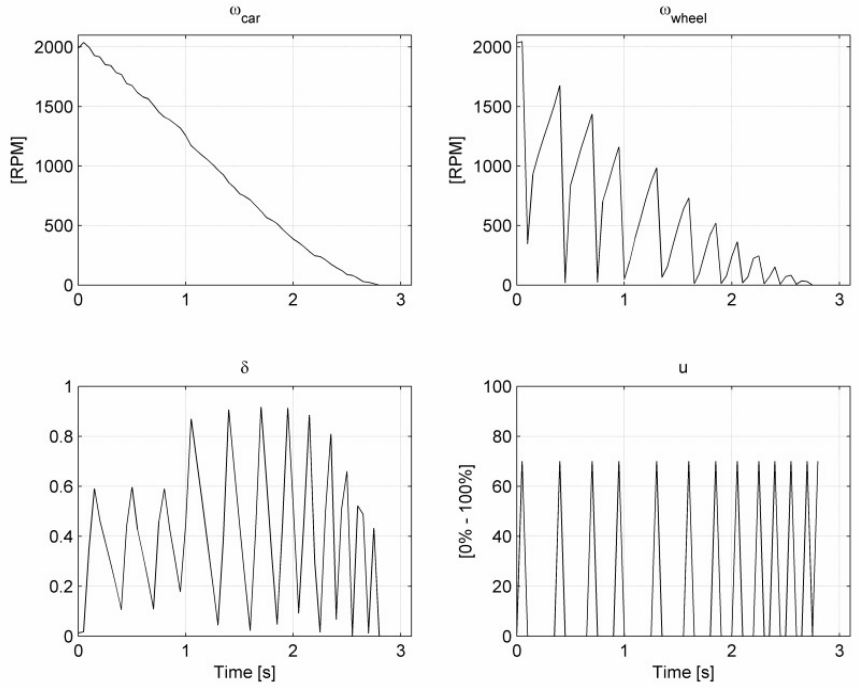

Fig. 7. Braking experiment.
Executed in MATLAB/Simulink/RTW/RTWT environment

The braking experiment performed in the MABLAB/Simulink/RTW/RTWT real-time environment is shown in Figure 7. In this environment, the jitter of the timer does not exceed a few microseconds [6]. One can notice the changes of the wheel speed ω_{WHEEL} caused by the control signal u, leading to stopping the car ω_{CAR}. At the same time slip δ is kept outside the locking range of the wheel.

4.2. Architecture of the control system

The device acting as the ABS system is a mobile phone Samsung Galaxy S II equipped with dual-core ARM Cortex A9 processor, running at 1.2 GHz with 1GB RAM memory.

Figure 8 shows the architecture of the control system. Also the view of the control application is given in the figure. Communication with ABS model is implemented via a wireless network through a wireless router. The PC computer equipped with the measurement and control I/O board is connected to the router by an Ethernet cable. The only PC's role is to transfer the measurement and control signals. The PC acts as a TCP/IP server and operates as a bridge between the mobile phone and the ABS model.

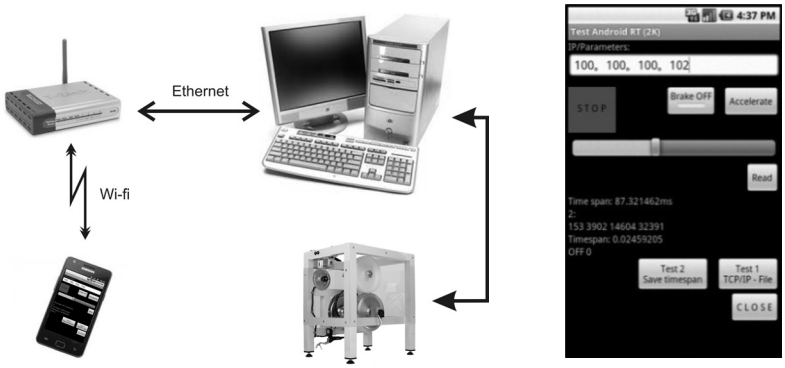

Fig. 8. Architecture of the control system and view of the control application

The control application on the phone is created entirely in Java. The main task of the application is to calculate the control signal every 50 ms. It also supports GUI interface, TCP/IP communication and data acquisition functions. During each sampling period the application requests and reads measurements from the PC, calculates and sends the control signal to the PC and performs data acquisition functions.

4.3. Results of the experiment

The result of the experiment, where the mobile phone operates as a controller, is shown in Figure 9. The phone runs a relay controller to stabilise the slip value with parameters similar to the experiment presented in Figure 7. In both cases the related time diagrams are similar. The Android device works as a system controller for fast ABS model.

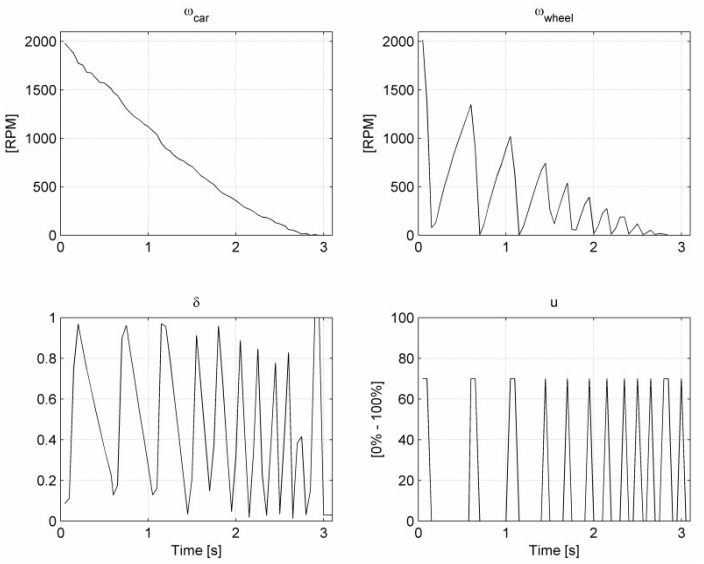

Fig. 9. Braking experiment. ABS model controlled by Android device

Figure 10 shows the measured timer punctuality ΔT and the time required by the controller to calculate the control signal (as the difference between the entry time t_{end} and leave time t_{begin} of the controller function). One can notice the jitter and variable execution time of the controller. The execution time doesn't depend on the algorithm. It is disturbed mainly by the transfer time through a wireless TCP/IP channel. It should be noted that the data presented in Figure 10 is one of the worst obtained during the experiments. In particular, the observed jitter values were generally much smaller than presented.

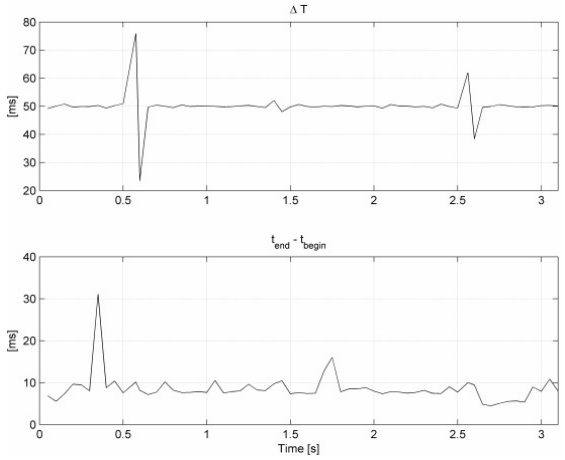

Fig. 10. Punctuality and execution time of the ABS controller

5. Conclusions

The source codes of the Android OS are available. This opens the way to modify the system to support the requirements of the real-time systems [7]. It seems to be possible to replace the current Linux kernel to kernel extensions including real-time functions. Also the Dalvik machine can be enrich with real-time capabilities. As a third option a dual OS configuration, where Linux coexists with a real-time supervisor, can be considered.

This paper focuses only on the testing for real-time applications some selected features of the unmodified Android operating system. The results of the experiments confirm the possibility of the use of small portable devices for real-time control. It is demonstrated practically the ability to work in real-time mode with Java applications.

Experiments have been completed successfully, but taking into account the presented results it is difficult to argue that the mobile phone is a recommended device to control a real car braking system. Punctuality of the time events differs from expectations of *hard real-time* systems, placing the presented solution in the *soft real-time* area. During the experiments only the laboratory model was used, for which some unpunctuality does not result in serious consequences.

It remains an open case the area of application of the presented solutions. Portable devices equipped with Android can benefit from a richness of features of the OS. They are able to use a high level Java language and can be attractive for systems requiring up to several controls per second. As an example of applicable target plants service equipment can be considered, which operates in a manual mode or performs only simple control sequences. Small size, the ability to authorize the devices and thus assign to each piece individual rights, is important advantages in this case.

References

[1] http://thenextweb.com/mwc/2012/02/27/android-growing-250-year-on-year-with-over-300-million-total-devices-worldwide/
[2] http://developer.android.com/guide/basics/what-is-android.html
[3] *Android Open Source Project*. http://source.android.com
[4] Ash M., *TCP/IP Sockets in Java*. Morgan Kaufman, 2002.
[5] *ABS. The laboratory Anti-lock Breaking System. User's Manual*. Dokumentacja techniczna firmy InTeCo Sp. z o.o., Kraków, Polska.
[6] Kołek K., Turnau A., *FPGA as a part of MS Windows control environment.* Computer Science, vol. 8, spec. ed., 2007, pp. 61–68.
[7] Maia C., Nogueira L.M., Pinho L.M., *Evaluating Android OS for Embedded Real-Time Systems*. http://www.ittc.ku.edu/~niehaus/classes/753-f10/documents/Android_RealTime.pdf