Smartphone based sensor data collection system for earth and space applications

Prepared by:
Karl Kangur

Checked/Approved/Signed by:
Anton Ivanov
Federico Belloni

SwissSpace Center
EPFL - Lausanne
Switzerland
January 20, 2014

Record of revisions

ISS/REV	Date	Modifications	Created/modified by
1/0	December 20, 2013	First draft	Karl Kangur
1/1	December 30, 2013	Second draft	Karl Kangur
1/2	January 7, 2014	Added section about PhoneSats, other minor modifications	Karl Kangur
1/3	January 11, 2014	Complete version	Karl Kangur

Contents

Terms and abbreviations

GPIO General Purpose Input Output

I^{2}C Inter-Integrated Circuit

TWI Two Wire Interface

UART Universal Asynchronous Receiver/Transmitter

SPI Serial Peripheral Interface

API Application Programming Interface

NFC Near Field Communication

IDE Integrated Development Environment

MCU Microcontroller Unit

GCC GNU Compiler Collection

TCP Transmission Control Protocol

SDK Software Development Kit

RISC Reduced Instruction Set Computing

ALU Arithmetic logic unit

1 Introduction

1.1 PhoneSat projects

PhoneSat projects aim to use commercial off-the-shelf (COTS) parts to build a satellite. The idea being to use the recent mass produced (and thus cheap), highly integrated phone technology instead of developing expensive in-house electronics in small satellites. It turns out current smartphones can function well in the vacuum, radiation and extreme temperature cycles of space and thus are suited for short missions in near-earth orbit.

In 2013 five successful PhoneSat projects have been launched to space: STRaND-1 [1], Alexander, Graham, Bell and PhoneSat 2.4 [2]. Next iterations are already programmed to be launched in February 2014. Another PhoneSat project named *Project BLAST* [21] also aims to use a smartphone as on-board computer, but it has only been ballon launched.

1.2 Goals

The primary goal of this project is to interface a smartphone with a low-level device without modifying it. The low-level device can be modified and should be able to exchange data with other peripherals such as sensors. The idea is to use the smartphone as a computer and wireless communication device and a the low-level device to forward data from sensors to the smartphone or control actuators.

The ultimate goal is to launch a PhoneSat project into earth's orbit using the smartphone as a cheap, powerful embedded system for the on-board computer and use the low-level device for things such as talking to sensors, actuators and other devices on the satellite, this projects brings the interface between the smartphone and external devices.

1.3 Smartphone

The reason why a modern smartphone was chosen for this project is mainly because of the highly integrated nature: the combination of an array of very interesting sensors, in a compact form all while using very little power makes it a perfect candidate for a satellite computer. Moreover smartphones are cheap and much more available compared to space grade electronics.

The decision to use an Android phone over other competitors in the smartphone market was mainly because it's the only system to support USB host mode, that is it can talk to any external USB device that is connected to its USB port. Other factors were its openness, good documentation and the extremely wide choice of hardware on the market.

One main advantage a phone operating system has over a more conventional one is that it manages its memory as efficiently as possible to keep power consumption at a minimum since phones are usually battery-powered.

Another advantage that makes the Android system suitable for this project is its community: it has accumulated many contributors from all over the world whom are willing to help and share their knowledge making it easy to find help for specific questions. Websites such as Stack Overflow, Android Enthusiasts, XDA-developers and many others are a great source of help.

1.4 Arduino

Arduino is a popular electronics prototyping platform. Arduino products use Atmel micro controllers, they are low-level devices with very interesting hardware features such as I^2C, SPI, UART and USB communication. Basically Arduino can be viewed as a simple C layer that makes micro controller programming easy for beginners and lowers development time.

An Arduino board is basically a little more than a breakout board for an Atmel micro controller. As with Android a huge community is behind Arduino making projects, programs and libraries for all sorts of devices. It can be programmed in C, C++ or even in assembly, contrary to Android there is no operating system or other underlying software besides the user written program.

The reason to use Arduino in this project was mainly because of its popularity, the existing software, libraries and the ever so expanding availability of new Arduino compatible hardware make it a fast to integrate and long-term proof platform. All Arduino products are open-source, the files for the printed circuit boards and source code are available on their website [3].

Other products such as the Raspeberry Pi [4] were considered, but ultimately the Arduino prevailed as the best solution because of its price, power consumption and community.

2 Physical framework

2.1 Overview

The physical framework consists of an Android phone, an Arduino board and other low-level devices such as sensors. The phone and the Arduino board are connected via USB with micro-USB to micro-USB cable. Other devices are attached to the Arduino via its general purpose input output (GPIO) ports with electrical wiring.

The Android phone used in this project is the *Galaxy Nexus* [17], it was mainly chosen because it was the most readily available. The Galaxy Nexus has been codeveloped by Google and Samsung, Google has a policy to stay away from bloatware which means that there isn't a manufacturer overlay or proprietary special features. It represents Android phones well as any application developed for it should work on all other Android phone models.

The Arduino model used in this project is the *Arduino Leonardo* [5], or rather a 100% compatible copy of it [7]. It has built-in USB connectivity which is the most important feature as it's the only way to connect it to the smartphone without opening it and tapping into lower level communication methods. It was chosen because is was most readily available, technically all Arduino products are compatible with this project with little to no modifications to the source code.

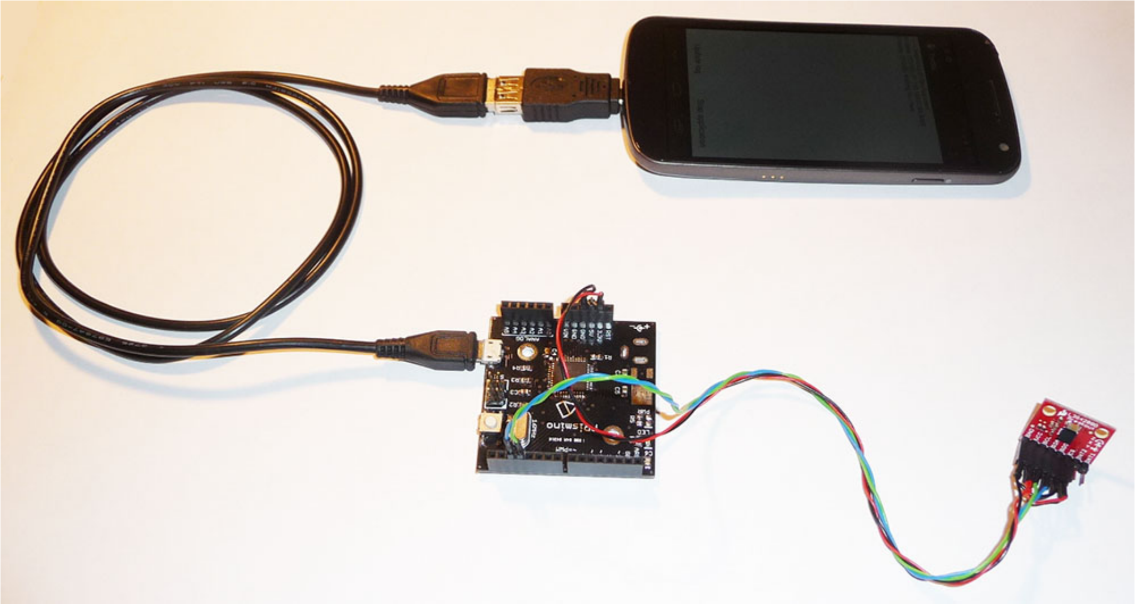


Figure 1: The actual physical framework setup: the Android phone is connected to the unofficial Arduino via a USB On-The-Go (OTG) cable and the Arduino board is connected to a gyroscope (L3G4200D) with power and data lines.

The figure 2 shows the system flowchart, there are two ways to communicate with the smartphone, more details in section 3.4.

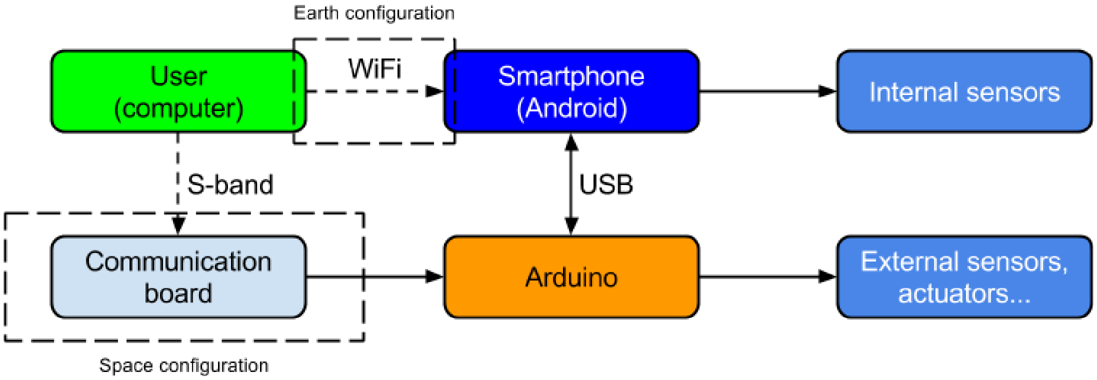


Figure 2: System flowchart, depending on the configuration the user can either connect via WiFi or a communication board.

2.2 Technical specifications

A short list of the most important technical specifications for both systems is shown on table 1. The Android smartphone has a lot more built-in communication interfaces, but since in this project the phone was not modified these interfaces couldn't be accessed, moreover using these interfaces would've required very low-level programming which is hard, if not impossible, to do because of security restrictions and it would've restricted the project to this particular Android phone model.

	Android	Arduino
Model	Galaxy Nexus	Leonardo
CPU	OMAP4460	ATmega32U4
Speed	1.2GHz	16MHz
Operating system	Android 4.3	-
Wired comm.	USB	USB, GPIO
Wireless comm.	WiFi, Bluetooth, NFC, 3G	-
Idle power consumption	40mW	150mW[1], 1mW[2]

Table 1: Summary of Android smartphone and Arduino micro controller technical specifications

2.3 Sensors

The Android phones have some interesting internal sensors (accelerometer, compass, gyroscope...), they can be accessed at will by the application, the low-level interface is taken care of by the operating system and the user can simply ask the program to sample them.

[1] Normal use

[2] Power-down mode

3 Android Application

3.1 Overview

The Android application programmed for this project adheres as much as possible to the official Android API guidelines given by Google [8]. This theoretically allows the application to be run on any Android phone, some phones with less features might not be capable though. The application requires Android 4.3 (Jelly Bean) operating system, all newer Android operating systems are retro-compatible with older applications.

The phone's operating system makes a very good use of the hardware and minimises power consumption for obvious reasons, when the screen is not used the front-end of an application is automatically stopped, but background services can continue running, this is why the application works as a background service.

3.2 Programming environment

The Android application is programmed with Eclipse, a powerful IDE, with the Android software development kit add-on [9] that allows it to compile and upload applications to an Android smartphone. This IDE comes with very useful debugging tools and it displays all important information about the phone's activities when it's connected via the USB cable to the computer. More importantly it can detect memory leaks and give constructive feedback on application crashes.

3.3 Approach

The Android applications are composed of different types of activities [10] and services [11], some meant for the user to interact with the application and others intended to work on background processes.

The application workflow was inspired from STRaND project [1]. The idea is to divide the application in different services, running only those that are needed, the main argument being power consumption and application robustness. A service can malfunction and be restarted without stopping the whole application. The main service runs indefinitely and distributes tasks to sub-services.

If no user is connected to the phone (via WiFi, Bluetooth, USB...) the application will shut down all non essential services and will wait for a client to connect to it.

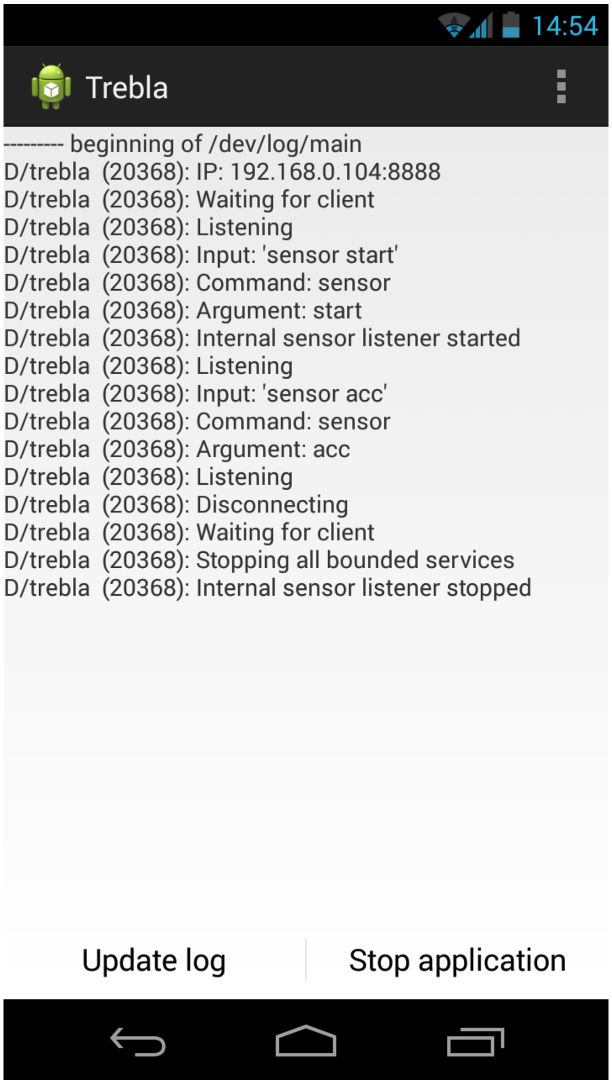
3.4 Application configurations

The application has two configurations: sensor data sampling and a space oriented configuration like shown on figure 2, while on earth the internal communication methods such as WiFi and Bluetooth can be used, while in space those methods cannot be used.

3.4.1 Sensor data sampling

Since the smartphone has some interesting built-in communication methods they can be used to make a sensor logger or transmitter out of it. Current commercial systems for data logging are expensive, bulky and often run on proprietary software having less freedom and control.

The sensor data sampling configuration programmed for this application uses the phone's WiFi connection over TCP socket making is accessible from anywhere as long as it's on the same network as the user computer. The application creates a socket server on the phone that waits for clients to connect, once



```
--------- beginning of /dev/log/main
D/trebla  (20368): IP: 192.168.0.104:8888
D/trebla  (20368): Waiting for client
D/trebla  (20368): Listening
D/trebla  (20368): Input: 'sensor start'
D/trebla  (20368): Command: sensor
D/trebla  (20368): Argument: start
D/trebla  (20368): Internal sensor listener started
D/trebla  (20368): Listening
D/trebla  (20368): Input: 'sensor acc'
D/trebla  (20368): Command: sensor
D/trebla  (20368): Argument: acc
D/trebla  (20368): Listening
D/trebla  (20368): Disconnecting
D/trebla  (20368): Waiting for client
D/trebla  (20368): Stopping all bounded services
D/trebla  (20368): Internal sensor listener stopped
```

Figure 3: A screenshot of the application front end, it displays the Android debug log of the application.

the client is connected he can interact with the application like a command line prompt (see Instructions section).

Other communication methods such as Bluetooth or data over 3G could technically also be used, but they weren't implemented in the course of this project.

3.4.2 Space oriented

Since there is no signal in space and built-in methods (3G, Bluetooth, WiFi) do not work another configuration had to be studied. The idea is the use the Arduino as input: instead of communicating with it to get sensor data the smartphone gets the user input from the Arduino which gets it from a communication board that can talk with a base station on earth.

In space configuration the phone must receive power form somewhere, but in host mode it sources power for the connected USB device, the USB port is the only charging input and it cannot be used in this case. One solution would be to use the very accessible battery connections to deliver power to the phone. It might also be more efficient to use an external supply for the USB device instead of relying on the phone's internal regulator.

3.5 Communication services

3.5.1 WiFi

The most straight forward way to exchange data with a computer wirelessly over short distances is via WiFi: a service makes a socket server on the phone that waits for a client to connect. The client must know the IP of the phone and must be connected to the same network. Once the application is launched it displays its IP on screen.

3.5.2 Bluetooth

Bluetooth allows for a peer-to-peer communication, albeit with limited range, to send and receive data. It has lower bandwidth than WiFi, but in most sensor sampling applications is should be fast enough. It was not implemented in the scope of this project, but it deserves a mention for eventual future developments.

3.5.3 USB

The USB communication is the only available wired connection on the smartphone, it is used to load the application, debug and in the scope of this project to communicate with the external peripheral.

The Android smartphone can be programmed to act as a USB host: it can supply power at USB levels (5V) and announce itself to the connected peripheral as a computer. That means the Arduino, and everything connected to it, is powered by the battery of the smartphone.

To use the USB device the Android phone has to know how to connect and interface with it: it needs a driver. There are two ways to solve the driver problem: recompile the phone kernel including all the necessary drivers or program a "soft driver" in the application. The second method was selected as it required less time to be implemented. An open-source project [20] was used as reference for this part of the application.

This single USB port might be expanded with a USB hub allowing multiple devices to be connected to be smartphone.

There is currently another PhoneSat project in works [21] using the same smartphone as this project, the phone/external device interface also uses the USB port, but instead of using the official API it uses actual drivers and kernel modifications making it more difficult to port on another hardware compared to this project.

3.6 Internal sensors

The Galaxy Nexus has a very interesting array of sensors, as shown on table 2, that could be useful in space such as geomagnetic sensor, accelerometer and gyroscope for attitude control. More recent phones will probably feature more accurate and varied sensors.

All the internal sensors can be easily read by the application and values forwarded to the user or used internally. Pre-processing is done at operating system level (transforming sensor output values to real

Sensor	Make	Max. range	Max. resolution
Accelerometer	BOSCH BMA250	±16g	3.9mg
Gyroscope	InvenSense MPU3050	$\pm 2000^o/s$	$0.0038^o/s$
Proximity	Sharp GP2A	10 to 80 cm	Analog readout
Barometer	BOSCH BMP180	300 to 1100 hPa	-4.0 to +2.0 hPa
Geomagnetic	Yamaha YAS530	±800uT	0.15uT (X,Y), 0.3uT (Z)

Table 2: Technical specifications of the Galaxy Nexus smartphone internal sensors

values) so that if another phone than the Galaxy Nexus is used with other sensors the values read by the application will hold true without any modifications to the source code.

There are two cameras on this smartphone, but it is complicated to use them because of the nature of the application: for security and privacy reasons it is not possible to take pictures or video without showing a preview on screen, but the application is meant to run using services that do not have access to the front end (screen). When the screen is off all activities that are front-end based are paused, only back end services are allowed to continue to run. There is probably a way to still use the cameras and comply with the security measures by activating the screen to take a picture, but it's very convoluted. It wasn't implemented in the course of this project, but the code base for the camera exists.

During gyroscope testing on a rotation table it was discovered that when the phone's rotation speed is held constant for about 20 seconds the returned gyroscope value adopts this constant offset as a new bias, this is quite annoying and means the internal gyroscope cannot be used unless this bias reset can be prevented or raw values can be read without the system's preprocessing.

3.7 Instructions

The application works like a command line prompt, it receives text strings, parses them, recognises commands, executes them and finally replies with the right data. Table 3 summarises the available commands, how to use them and return values.

Refer to the official documentation [19] for more information on the values returned by the sensors.

While communicating with the USB device the replies are asynchronous, that is once the data is sent it may take a while to reply, other tasks such as internal sensor polling may be executed meanwhile, but that might produce inconsistent replies, one must make sure what the reply from the application was for.

4 Arduino

4.1 Overview

Arduino products use Atmel micro controllers, they are small and cheap (usually less than 5CHF). The one chosen for this project was the *Arduino Leonardo* [5] that uses the ATmega32U4 [12] micro controller and runs at 16MHz, it was chosen because it features built-in USB connectivity and it was the most readily available.

Since the *Arduino Leonardo* is essentially a breakout board for the micro controller with a specific pinout we don't actually need the product itself, the really interesting and useful thing about Arduino is the boot-loader [6] and libraries: there are libraries for almost everything thanks to the Arduino community. Without the boot-loader one has to use a special programmer to load the compiled program

Instuction	Parameters	Description
sensor	start	Starts sensor sampling service, samples all internal sensors. Without this the internal sensors cannot be sampled.
sensor	stop	Stops sensor sampling service.
sensor	acc	Accelerometer data in `[<x>,<y>,<z>]` format, units in m/s^2.
sensor	gyro	Gyroscope data in `[<x>,<y>,<z>]` format, units in rad/s.
sensor	temp	Temperature value in `[<temperature>]` format in Celsius. *
sensor	pressure	Absolute pressure value in `[<pressure>]` format, units in hPa (millibar).
sensor	proximity	Proximity sensor value in `[<proximity>]` format in centimeters.
sensor	light	Light sensor value in `[<light>]` format, units in lux.
sensor	mfield	Magnetic field data in `[<x>,<y>,<z>]` format, units in micro-Tesla (μT).
sensor	rh	Relative humidity value in `[<rh>]` format in percent. *
sensor	rotation	Rotation values which are filtered values using the compass, accelerometer and gyroscope, refer to the official documentation [19] for more information for these values.
usb	connect	Establish a connection with the external connected USB device.
usb	-list	List all connected USB devices with all their attributes, only one device can be connected at a time so only one device will be shown when connected.
usb	-w [data]	Send data to the USB device, the device must be connected with `usb connect` beforehand. It returns the number of transferred bytes.
usb	-r	Starts the asynchronous USB reading thread. Needed to receive data from the USB device. All incoming data is automatically forwarded to the connected client.
battery	start	Starts the battery sensor sampling service.
battery	stop	Stops battery sensor sampling service.
battery	state	Battery state in `[<voltage>,<temperature>,<charge>]` format, the charge is in percent.

Table 3: List of instructions. The *Galaxy Nexus* does not have all these sensors and will return a null value for the missing ones marked with an asterisk.

into the flash memory called an *in-system programmer* (ISP), the boot-loader is loaded only once and then the flash memory can be changed via the USB interface.

The board used in this project isn't an official Arduino product, but one that is 100% compatible. It was developed for a different project [7], but was ideal for this one.

Another very useful feature of the Atmel chips is that they can enter a power-down mode where they consume only a fraction of the power compared to normal operation mode. In this mode most of the micro controller features are shut down, it can be waken up with a watchdog timer, with internal interrupts (timeout) or with external interrupts (communication request, button press, sensors, external watchdog...).

4.2 Characteristics

Atmel micro controllers are programmed in C, C++ or assembly. They have a 10-bit analog-to-digital conversion module, not all pins can actually do it, the parenthesis in the GPIO column in table 4 show

Model	MCU	USB	GPIO (analog)	Clock	Voltage
Mini	ATmega328	No	14 (8)	16MHz	5V
Leonardo	ATmega32U4	Yes	20 (12)	16MHz	5V
Uno	ATmega328	No	14 (6)	16MHz	5V
Mega	ATmega2560	No	54 (16)	16MHz	5V
Due	AT91SAM3X8E	Yes	54 (12)	84MHz	3.3V

Table 4: Technical specifications of different Arduino products, see [13] for more information.

how many of the GPIO pins can also do ADC, the specifics are described in the data sheet of the micro controller.

Atmel micro controllers have also timers, these are asynchronous counters that can trigger events, they can be used to call some scripts regularly. Some have used them to implement pseudo-multitasking in a real time operating system that essentially distributes processing time between different scripts [18].

4.3 Communication

The ATmega32U4 has one very useful feature: integrated USB connectivity, this means it doesn't need a USB to serial chip. The USB stack is implemented by the Arduino boot-loader, it's completely transparent to the user who can send data to the host with a simple function.

The boot-loader registers the micro controller as a USB communications device class (CDC) for the host (smartphone), this is interpreted from the Android side in the application and the communication is established.

4.4 Sensors

The ATmega32U4 doesn't have any interesting integrated sensors, a badly calibrated temperature sensor ($\pm 10^oC$) is there for frequency compensation when the internal RC oscillator is used, the board in this project uses an external 16MHz quartz cristal for better accuracy.

The GPIO pins are really useful to read any kind of sensor, on the ATmega32U4 hardware implementation of different standard communication protocols exist: I^2C, UART, SPI and any other protocol can easily be implemented via bit-bang. The advantage of Arduino is that being so popular the community has developed a huge number of libraries, there's a high chance of finding a library for a sensor found on the market. Other interfaces can be implemented via bit-banging.

The one important thing to look out for is the logic levels: the Arduino used in this project runs at 5V, it can also run at 3.3V, but only at a maximum of 8MHz, this is defined by the boot-loader and fuse bits at a very low level. A sensor that cannot accept 5V as input must have level-shifters or a voltage divider in between it and the MCU pins.

In the course of this project an I^2C gyroscope running at 3.3V was used, usually internal pull-ups of the Arduino MCU are used for the clock and data lines, but as they are pulled to 5V they could damage the sensor, so they were disabled. Fortunately the sensor had its own pull-up resistors that pulled the line to 3.3V, the MCU was still able to communicate with the sensor flawlessly.

4.5 Program

The Arduino products are programmed with Arduino IDE [3]. The compiler is a port of GCC named AVR-GCC, it allows code to be compiled for Atmel micro controllers, to upload the compiled code to the MCU Arduino uses AVRDUDE, an open source utility. One can use any code editor as long as one compiles with AVR-GCC and uploads with AVRDUDE, but for simplicity sake Arduino IDE was used.

Atmel micro controllers are usually programmed with Atmel Studio [14], the official tool to program them, but it needs a physical programmer to load programs in the micro controller. A plug-in exists for Atmel Studio that allows it to program Arduino products via the boot-loader, without the programmer. It's a very powerful tool allowing virtual execution of the program, but it's for Windows only and all its tools are not necessary for development.

The program for this particular project is aimed at receiving commands, processing them and replying, for example when it receives the command ”g” it will request a sample from the gyroscope and once finished reply with the values. The program is short, simple and can be modified and uploaded fast, thanks to all this the developing time for the Arduino is extremely short.

One must know that given the simple RISC architecture of the micro controller it cannot handle float values very well, it's not implemented in the ALU and so operations on float values are extremely expensive, this is why no pre-processing is done when values are returned from sensors, all values returned by the program are raw and must be transformed to real values.

5 Interface

In order for one to use this project in future applications some example scripts were written to show how to communicate with the smartphone from a computer in two different ways via TCP socket: using Matlab and a script written in Python programming language.

5.1 Real-time sensor sampling and display in Matlab

To view sensor output in real time Matlab can establish a TCP socket connection with just a couple of lines, this makes it easy to build a graphical display of sensor values such as the one shown on figure 4. The source code is in the annex.

5.2 Python graphical terminal

To establish a connection from a computer to the smartphone via TCP socket a small terminal-like program with a graphical front-end shown on figure 5 was made in Python. Python being a very flexible and easy to use language it only took a couple of lines of code to build a socket-based communication. Then some input fields and buttons were added for ease of use. The source code is in the annex.

6 Testing

A proof of concept test was done on a rotation table while the internal and an external gyroscope sensors were sampled and compared. The test was successful and live measurements could be seen via a Matlab script as seen on figure 6. This is a typical application where cable routing is difficult and a wireless system with on-board power supply can be useful.

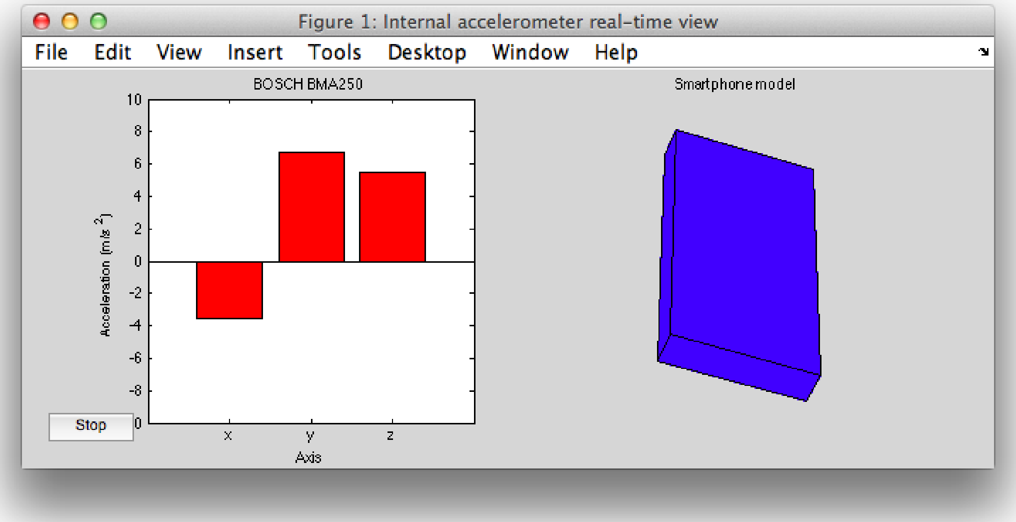
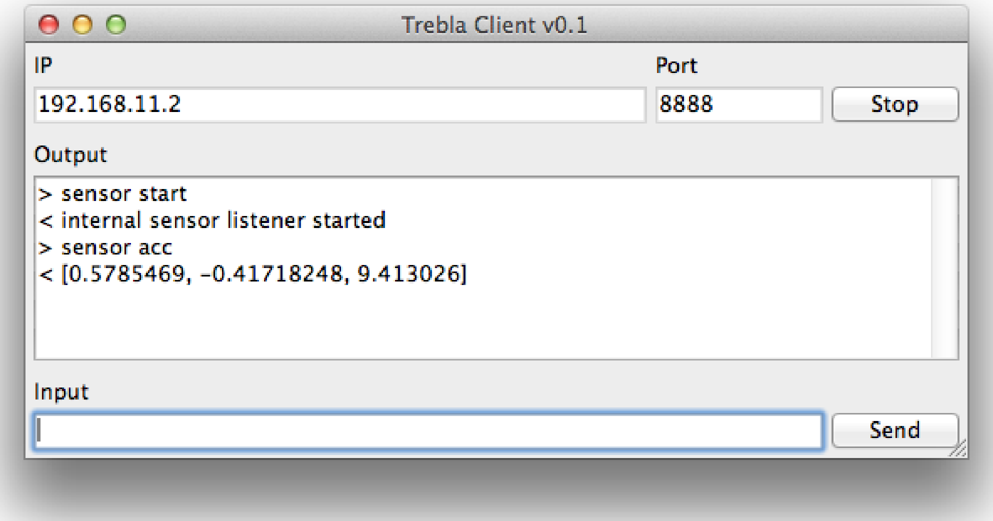


Figure 1: Internal accelerometer real−time view
File Edit View Insert Tools Desktop Window Help
BOSCH BMA250
Acceleration (m/s^2)
10
8
6
4
2
0
-2
-4
-6
-8
0
x
y
z
Axis
Stop
Smartphone model

Figure 4: Matlab script to display real time accelerometer values on a computer.



IP: 192.168.11.2

Port: 8888

Output

```
> sensor start
< internal sensor listener started
> sensor acc
< [0.5785469, -0.41718248, 9.413026]
```

Input

Figure 5: Program for establishing TCP socket connection between a computer and the smartphone application with graphical user interface.

As mentioned in the section 3.6 the smartphone internal gyroscope has a bias reset after about 20 seconds, probably to compensate for drift, this means if the phone has a constant rotation for more than 20 seconds the internal gyroscope sensor will indicate zero angular speed, even if it's false. The external gyroscope returns raw values and thus does not have this issue, on the figure 6 the saturation value of the external sensor is $250^o/s$, this is because it was configured that way, but it can go up to $2000^o/s$, this is modified in the Arduino code (appendix B) that sets up the external gyroscope sampling settings.

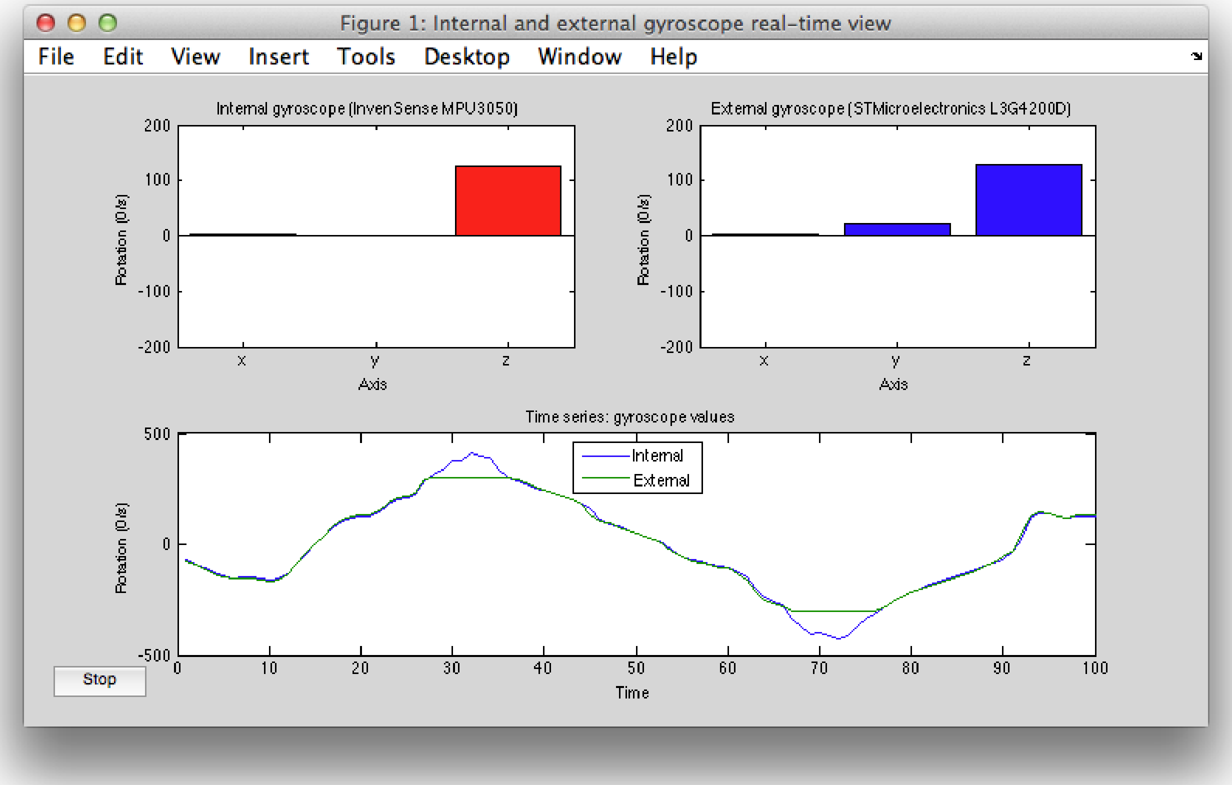


Figure 1: Internal and external gyroscope real–time view
File
Edit
View
Insert
Tools
Desktop
Window
Help
Internal gyroscope (InvenSense MPU3050)
External gyroscope (STMicroelectronics L3G4200D)
Rotation (0/s)
200
100
0
-100
-200
x
y
z
Axis
Time series: gyroscope values
Internal
External
500
0
-500
0
10
20
30
40
50
60
70
80
90
100
Time
Stop

Figure 6: Matlab script to display real time gyroscope values on a computer.

No other tests were performed, but it remains to be seen if the application is robust enough to handle all scenarios, space being an unforgiving place the system needs to be tested under the harsh conditions of space radiation, extreme temperature cycles and the effects on different internal components. The advantage of the current system is that the on-board computer (smartphone) can be tested separately from the rest of the system: while the smartphone is being tested the auxiliary systems can be out of the testing environment so eventual critical system failures could be pinpointed faster.

The application itself functions well, but needs more failsafes, currently it can handle some exceptions when things do not work out, but timeouts are not yet implemented, they are necessary to avoid hangups. For example when a client is lost the application might not always notice it, a timeout needs to be implemented so that the application would automatically disconnect from a client when no requests have been received within a minute or so.

The USB connection has not always be reliable, it worked most of the time, but on rare occasions the electrical contact was not good enough, a wiggle fixed the problem temporarily, but if it's to be used in space it must be soldered.

7 Conclusion

In the scope of this project an Android application was developed so a smartphone could be used for scientific research purposes and eventually used in a CubeSat project in space. The application uses the official Android API and no modifications were made to the phone making the application compatible with all Android phones. The external device used in this project (Arduino) can be replaced with any other device provided that it's either comparable with what has been implemented or some modifications to the application are needed, but the hardware can be modified without rendering this application completely obsolete.

In its current status the application is not quite ready for space applications which require lots of fail-safes and an actual project scope with well defined tasks, but it can be used for sensor sampling, the built-in features of the smartphone make this project interesting for a multitude of applications, being able to transfer and receive data with an external device makes the whole project very polyvalent.

The PhoneSat can drastically bring down the development of a satellite and it's only recently that people have started studying it seriously. In 2013 the first five PhoneSat projects were launched and all were successful, this project brings the software needed for smartphone integration to a satellite that can without a doubt benefit next generation low-cost satellites.

In very recent developments as of this writing Ubuntu is preparing its own operating system for Android mobile devices based on Linux [15]. This might bring a much more open and familiar system to smartphones and may make development of such projects much easier. It remains to be seen how it handles the limited power resources of a smartphone.

References

[1] C. P. Bridges, B. Yeomans, C. Iacopino, T. E. Frame, A. Schofield, S. Kenyon, M. N. Sweeting *Smartphone Qualification & Linux-based Tools for CubeSat Computing Payloads* 2013: Surrey Space Centre, University of Surrey, Guildford, Surrey, United Kingdom

[2] NASA Ames' Smartphone Nanosatellite *PhoneSat* http://www.phonesat.org

[3] *Arduino website* http://arduino.cc/

[4] *Raspberry Pi* http://www.raspberrypi.org/

[5] *Arduino Leonardo* http://arduino.cc/en/Main/arduinoBoardLeonardo

[6] *Arduino Bootloader* http://arduino.cc/en/Hacking/Bootloader

[7] Robopoly *PRismino* https://github.com/Robopoly/PRismino

[8] Android Developer *Android API* https://developer.android.com/guide/components/index.html

[9] Android Developer *Android SDK* https://developer.android.com/sdk/index.html

[10] Android Developer *Activity lifecycle* http://developer.android.com/training/basics/activity-lifecycle/starting.html

[11] Android Developer *Service* http://developer.android.com/reference/android/app/Service.html

[12] Atmel *ATmega32U4* http://www.atmel.ch/devices/ATMEGA32U4.aspx

[13] *Arduino Product line* http://arduino.cc/en/Main/Products

[14] *Atmel Studio 6* http://www.atmel.ch/microsite/atmel_studio6/

[15] *Ubuntu Touch* http://www.ubuntu.com/phone

[16] Wikipedia *Android* http://en.wikipedia.org/wiki/Android_(operating_system)

[17] Wikipedia *Galaxy Nexus* http://en.wikipedia.org/wiki/Galaxy_Nexus

[18] G. Di Sirio *Real time operating system for Arduino: NilRTOS* https://code.google.com/p/rtoslibs/

[19] Android Developer *Sensor Events* http://developer.android.com/reference/android/hardware/SensorEvent.html

[20] GitHub *USB Serial for Android* https://github.com/mik3y/usb-serial-for-android

[21] University of Southampton *Project BLAST* http://projectblast.co.uk

A Android application source code

A.1 Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.nurgak.trebla"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="18" />

    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />

    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />

    <application
        android:name=".TreblaApplication"
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
            </intent-filter>

            <meta-data
                android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
                android:resource="@xml/usb_device_filter" />
        </activity>

        <service android:name=".TreblaService" >
        </service>
        <service android:name=".services.SocketServerService" >
        </service>
        <service android:name=".services.BluetoothClientService" >
        </service>
    </application>

</manifest>
```

Listing 1: AndroidManifest.xml

A.2 Classes

```java
package com.nurgak.trebla.services;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.BatteryManager;
import android.util.Log;

public class BatteryBroadcastReceiver extends BroadcastReceiver
{
  float[] batteryState = new float[3];

  Context context;

  public BatteryBroadcastReceiver(Context context)
  {
    this.context = context;
  }

  @Override
  public void onReceive(Context context, Intent intent)
  {
    // voltage
    batteryState[0] = intent.getIntExtra(BatteryManager.EXTRA_VOLTAGE, -1) / 1000;
    // temperature
    batteryState[1] = intent.getIntExtra(BatteryManager.EXTRA_TEMPERATURE, -1) / 10;
    // charge in percent
    batteryState[2] = 100*intent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1)/intent.
        getIntExtra(BatteryManager.EXTRA_SCALE, -1);
  }

  public void start()
  {
    // start battery state listener
    IntentFilter filter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
    context.registerReceiver(this, filter);
    Log.d("trebla", "Battery receiver started");
  }

  public void stop()
  {
    // stop listening to battery state
    context.unregisterReceiver(this);
    Log.d("trebla", "Battery receiver stopped");
  }

  public String getBatteryState()
  {
    return java.util.Arrays.toString(batteryState);
  }
}
```

Listing 2: BatteryBroadcastReceiver.java

```java
package com.nurgak.trebla.services;

import java.io.IOException;
import java.io.OutputStream;
import java.util.UUID;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.util.Log;

import com.nurgak.trebla.BoundService;

public class BluetoothClientService extends BoundService implements Runnable
{
  static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

  BluetoothAdapter bluetoothAdapter = null;
  BluetoothSocket btSocket = null;
  OutputStream outStream = null;
  BluetoothDevice device = null;

  // this should not be hardcoded
  static String address = "14:10:9F:E8:06:99";

  @Override
  public void onCreate()
  {
    bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

    if(bluetoothAdapter == null || !bluetoothAdapter.isEnabled())
    {
      Log.d("trebla", "Bluetooth not enabled");
      return;
    }

    device = bluetoothAdapter.getRemoteDevice(address);

    try
    {
      btSocket = device.createRfcommSocketToServiceRecord(MY_UUID);
    }
    catch(IOException e)
    {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    // try to cancel bluetooth discovery just in case it’s running
    bluetoothAdapter.cancelDiscovery();

    new Thread(this).start();
  }

  @Override
  public void run()
  {
    while(true)
    {
      // blocking connection here, put in another thread
      Log.d("trebla", "Waiting for a client to connect");
      try
      {
        btSocket.connect();
      }
      catch(IOException e)
      {
        try
        {
          btSocket.close();
        }
        catch(IOException e1)
        {
          // TODO Auto-generated catch block
          e1.printStackTrace();
        }
      }

      try
      {
        outStream = btSocket.getOutputStream();
      }
      catch(IOException e)
      {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }

      // parse command here

      String message = "Hello message from client to server.";
      byte[] msgBuffer = message.getBytes();
      try
      {
        outStream.write(msgBuffer);
      }
      catch(IOException e)
      {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }
    }
  }
}
```

Listing 3: BluetoothClientService.java

```java
package com.nurgak.trebla.services;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;

import android.app.Service;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.IBinder;
import android.util.Log;

public class BluetoothCommunicationManager extends Service
{
  private final static String TAG = "trebla";
  public final static boolean D = true;

  BluetoothAdapter bluetoothAdapter;

  // Member fields
  private BluetoothThread bluetoothThread;
  private boolean busy, stoppingConnection;

  public String listBluetoothDevices()
  {
    bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

    // service needs a context, for this it needs to be started with startService or bindService
    // otherwise these calls will return null pointer errors
    registerReceiver(Receiver, new IntentFilter(BluetoothDevice.ACTION_FOUND));
    registerReceiver(Receiver, new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED));

    bluetoothAdapter.startDiscovery();

    return null;
  }

  private final BroadcastReceiver Receiver = new BroadcastReceiver()
  {
    @Override
    public void onReceive(Context context, Intent intent)
    {
      String action = intent.getAction();
      if(BluetoothDevice.ACTION_FOUND.equals(action))
      {
        // Found a device in range
        BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
        // If it's not a paired device add it to the list
        if(device.getBondState() != BluetoothDevice.BOND_BONDED)
        {
          Log.d(TAG, "Device found: " + device.getName());
        }
      }
      else if(BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action))
      {
        Log.d(TAG, "Finished discoverting devices");
      }
    }
  };

  /**
   * Start the ConnectThread to initiate a connection to a remote device.
   *
   * @param device
   *             The BluetoothDevice to connect
   */
  public synchronized void connect(BluetoothDevice device)
  {
    if(D)
      Log.i(TAG, "Connecting to " + device.getName());
    stoppingConnection = false;
    busy = false;

    // Cancel any thread currently running a connection
    if(bluetoothThread != null)
    {
      bluetoothThread.cancel();
      bluetoothThread = null;
    }

    // Start the thread to connect with the given device
    bluetoothThread = new BluetoothThread(device);
    bluetoothThread.start();
  }

  /**
   * This thread runs during a connection with a remote device. It handles the
   * initial connection and all incoming and outgoing transmissions.
   */
  private class BluetoothThread extends Thread
  {
    private final BluetoothSocket socket;
    private InputStream inStream;
    private OutputStream outStream;

    public BluetoothThread(BluetoothDevice device)
    {
      BluetoothSocket tmp = null;
      try
      {
        // General purpose UUID
        tmp = device.createInsecureRfcommSocketToServiceRecord(UUID.fromString("
            00001101-0000-1000-8000-00805F9B34FB"));
      }
      catch(IOException e)
      {
        e.printStackTrace();
      }
      socket = tmp;
    }

    public void run()
    {
      // Connect to the socket
      try
      {
        // Blocking function, needs the timeout
        if(D)
          Log.i(TAG, "Connecting to socket");
        socket.connect();
      }
      catch(IOException e)
      {
        // If the user didn't cancel the connection then it has failed (timeout)
        if(!stoppingConnection)
        {
          if(D)
            Log.e(TAG, "Cound not connect to socket");
          e.printStackTrace();
          try
          {
            socket.close();
          }
          catch(IOException e1)
          {
            if(D)
              Log.e(TAG, "Cound not close the socket");
            e1.printStackTrace();
          }
          disconnect();
        }
        return;
      }

      // Get the BluetoothSocket input and output streams
      try
      {
        inStream = socket.getInputStream();
        outStream = socket.getOutputStream();
      }
      catch(IOException e)
      {
        // Failed to get the streams
        disconnect();
        e.printStackTrace();
        return;
      }

      byte[] buffer = new byte[1024];
      byte ch;
      int bytes;
      String input;

      // Keep listening to the InputStream while connected
      while(true)
      {
        try
        {
          // Make a packet, use \n (new line or NL) as packet end
          // println() used in Arduino code adds \r\n to the end of the stream
          bytes = 0;
          while((ch = (byte) inStream.read()) != '\n')
          {
            buffer[bytes++] = ch;
          }
          // Prevent read errors (if you mess enough with it)
          if(bytes > 0)
          {
            // The carriage return (\r) character has to be removed
            input = new String(buffer, "UTF-8").substring(0, bytes - 1);

            if(D)
              Log.v(TAG, "Read: " + input);
          }
          busy = false;

        }
        catch(IOException e)
        {
          // read() will inevitably throw an error, even when just disconnecting
          if(!stoppingConnection)
          {
            if(D)
              Log.e(TAG, "Failed to read");
            e.printStackTrace();
            disconnect();
          }
          break;
        }
      }
    }

    public boolean write(String out)
    {
      if(outStream == null)
      {
        return false;
      }
```

```

      if(D)
        Log.v(TAG, "Write: " + out);
      try
      {
        if(out != null)
        {
          outStream.write(out.getBytes());
        }
        else
        {
          // This is a special case for the filler
          outStream.write(0);
        }
        // End packet with a new line
        outStream.write('\n');
        return true;
      }
      catch(IOException e)
      {
        e.printStackTrace();
      }
      return false;
    }

    public void cancel()
    {
      try
      {
        if(inStream != null)
        {
          inStream.close();
        }
        if(outStream != null)
        {
          outStream.close();
        }
        if(socket != null)
        {
          socket.close();
        }
      }
      catch(IOException e)
      {
        e.printStackTrace();
      }
    }
  }

  /**
   * This method sends data to the Bluetooth device in an unsynchronized
   * manner, actually it calls the write() method inside the connected thread,
   * but it also makes sure the device is not busy. If "r" is sent (reset
   * flag) it will pass all flags and will be sent even if the device is busy.
   *
```

```java
   * @param out
   *            String to send to the Bluetooth device
   * @return Success of failure to write
   */
  public boolean write(String out)
  {
    // The device hasn't finished processing last command, reset commands ("r") it always get sent
    if(busy && !out.equals(out))
    {
      if(D)
        Log.v(TAG, "Busy");
      return false;
    }
    busy = true;

    // Create temporary object
    BluetoothThread r;
    // Synchronize a copy of the BluetoothThread
    synchronized(this)
    {
      r = bluetoothThread;
    }
    // Perform the write unsynchronized
    return r.write(out);
  }

  /**
   * Stop all threads
   */
  public synchronized void disconnect()
  {
    // Do not stop twice
    if(!stoppingConnection)
    {
      stoppingConnection = true;
      if(D)
        Log.i(TAG, "Stop");
      if(bluetoothThread != null)
      {
        bluetoothThread.cancel();
        bluetoothThread = null;
      }
    }
  }

  @Override
  public IBinder onBind(Intent arg0)
  {
    // TODO Auto-generated method stub
    return null;
  }
}
```

Listing 4: BluetoothCommunicationManager.java

```java
package com.nurgak.trebla;

import android.app.Service;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Binder;
import android.os.IBinder;

/**
 * This class may be bounded to an application, activity or another service. Its
 * purpose is the serve as a generic boundable service to other services.
 * <p>
 * Some methods are useful only for the communication classes such as binding a
 * trebla service as a trebla service (which also extends this class in order to
 * be bound to a communication class) will never bind another trebla service
 * itself.
 */
public abstract class BoundService extends Service
{
  // instance of the local binder to pass to the client
  private final IBinder localIBinder = new LocalBinder();

  // service and connection to the bounded class
  protected TreblaService treblaService = null;
  private ServiceConnection treblaServiceConnection = null;

  @Override
  public IBinder onBind(Intent intent)
  {
    // must return an IBinder for this service to be bound to the main activity
    return localIBinder;
  }

  // this binder will return the enclosing BinderService instance.
  public class LocalBinder extends Binder
  {
    // return enclosing BinderService instance
    public BoundService getBoundService()
    {
      // required in order to bind the service to the activity/application/service
      return BoundService.this;
    }
  }

  @Override
  public int onStartCommand(Intent intent, int flags, int startId)
  {
    // this will keep this service alive no matter what, unless explicitly stopped by user/asked by application
    return START_STICKY;
  }

  public void bindTreblaService()
  {
    // bind a TreblaService instance to this service
    Intent intent = new Intent(this, TreblaService.class);
    treblaServiceConnection = new ServiceConnection()
    {
      @Override
      public void onServiceConnected(ComponentName name, IBinder binder)
      {
        // bounded service instance which can be used to call its methods directly
        treblaService = (TreblaService) ((BoundService.LocalBinder) binder).getBoundService();
      }

      @Override
      public void onServiceDisconnected(ComponentName name)
      {
        // TODO Auto-generated method stub
      }
    };
    bindService(intent, treblaServiceConnection, Context.BIND_AUTO_CREATE);
  }

  // this method is only used by the communication classes in order to unbind from their trebla service
  public void unBindTreblaService()
  {
    if(treblaServiceConnection != null)
    {
      // by unbinding all the trebla service clients (this is the only one) it will get stop the service
      unbindService(treblaServiceConnection);
      treblaServiceConnection = null;
      treblaService = null;
    }
  }

  @Override
  public void onDestroy()
  {
    // just in case unbind from a bounded trebla service if there is any
    unBindTreblaService();
    super.onDestroy();
  }
}
```

Listing 5: BoundService.java

```java
package com.nurgak.trebla.services;

import java.io.IOException;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.ImageFormat;
import android.hardware.Camera;
import android.hardware.Camera.PictureCallback;
import android.os.IBinder;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class CameraPicture extends Service implements PictureCallback, Camera.PreviewCallback
{
  SurfaceView dummySurfaceView;
  SurfaceHolder dummySurfaceHolder;
  Camera camera = null;

  // image bitmap data
  Bitmap bmp = null;

  byte[] pictureData = null;

  Context context;

  public CameraPicture(Context context)
  {
    //this.context = context.getApplicationContext();

    camera = Camera.open();

    Camera.Parameters parameters = camera.getParameters();
    parameters.setPictureFormat(ImageFormat.JPEG);

    camera.setParameters(parameters);
  }

  public byte[] getPictureData()
  {
    return pictureData;
  }

  public void takePicture()
  {
    // android needs to show a preview, so direct the preview to a dummy surface view
    dummySurfaceView = new SurfaceView(this);
    //dummySurfaceView = (SurfaceView) this.findViewById(R.id.cameraSurfaveView);
    //SurfaceTexture dummySurfaceTexture = new SurfaceTexture(1);
    dummySurfaceHolder = dummySurfaceView.getHolder();
    try
    {
      //camera.setPreviewDisplay(null);
      camera.setPreviewDisplay(dummySurfaceView.getHolder());
      //camera.setPreviewTexture(dummySurfaceTexture);
      camera.setPreviewCallback(this);
      camera.startPreview();
    }
    catch(IOException e)
    {
      e.printStackTrace();
    }

    Log.d("trebla", "Taking a picture");

    // take a picture and call an event once the data is available
    camera.takePicture(null, null, this);

    Log.d("trebla", "Delaying");

    camera.release();
  }

  @Override
  public void onPictureTaken(byte[] data, Camera camera)
  {
    // decode the data obtained by the camera into a bitmap
    pictureData = data;
    bmp = BitmapFactory.decodeByteArray(data, 0, data.length);
  }

  @Override
  public void onPreviewFrame(byte[] data, Camera camera)
  {
    pictureData = data;
    Log.d("trebla", "Got a preview frame");
  }

  @Override
  public IBinder onBind(Intent arg0)
  {
    // TODO Auto-generated method stub
    return null;
  }
}
```

Listing 6: CameraPicture.java

```
package com.nurgak.trebla.services;

import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import android.util.Log;

public class Command
{
  public String command = "";
  public String argument = "";
  Map<String, String> flags = new HashMap<String, String>();

  Pattern partsPattern = Pattern.compile("^([^\\s]+)(?:\\s+-.+?)?\\s*?([^\\s-]+)?$");
  Pattern paramsPattern = Pattern.compile("-{1,2}([^\\s=]+)(?:=([^\"\\s]+|\"[^\"]+\"))?");
  Matcher partsMatcher, paramsMatcher;

  public Command(String input)
  {
    if(input == null || input.length() == 0)
    {
      Log.d("trebla", "Null input detected");
      return;
    }

    partsMatcher = partsPattern.matcher(input);

    if(!partsMatcher.find())
    {
      Log.d("trebla", "Command did not match pattern");
      return;
    }

    command = partsMatcher.group(1);
    Log.d("trebla", "Command: " + partsMatcher.group(1));

    // commands do not have to have an argument
    if(partsMatcher.group(2) != null)
    {
      argument = partsMatcher.group(2);
      Log.d("trebla", "Argument: " + partsMatcher.group(2));
    }

    // flags
    paramsMatcher = paramsPattern.matcher(input);
    while(paramsMatcher.find())
    {
      flags.put(paramsMatcher.group(1), paramsMatcher.group(2));
      Log.d("trebla", "Param: " + paramsMatcher.group(1) + " = " + paramsMatcher.group(2));
    }
  }

  public String getFlag(String key)
  {
    if(flags.containsKey(key))
    {
      return flags.get(key);
    }
    return null;
  }

  public boolean checkFlag(String key)
  {
    if(flags.containsKey(key))
    {
      return true;
    }
    return false;
  }
}
```

Listing 7: Command.java

```java
package com.nurgak.trebla.services;

import java.util.List;

import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.Log;

public class InternalSensorListener implements SensorEventListener
{
  float acc[] = new float[3];
  float gyro[] = new float[3];
  float temp[] = new float[1];
  float pressure[] = new float[1];
  float gravity[] = new float[1];
  float light[] = new float[1];
  float mfield[] = new float[1];
  float rh[] = new float[1];
  float rotation[] = new float[1];
  float proximity[] = new float[1];

  SensorManager sensorManager = null;

  Context context;

  public InternalSensorListener(Context context)
  {
    // save context of the calling trebla service
    this.context = context;
  }

  public void start()
  {
    // internal sensor listener setup
    sensorManager = (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);

    // listen to all sensors
    List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
    for(Sensor sensor : sensors)
    {
      sensorManager.registerListener(this, sensor, SensorManager.SENSOR_DELAY_NORMAL);
    }
    Log.d("trebla", "Internal sensor listener started");
  }

  public void stop()
  {
    // unregister the listener for all sensors
    sensorManager.unregisterListener(this);
    sensorManager = null;
    Log.d("trebla", "Internal sensor listener stopped");
  }

  @Override
  public void onAccuracyChanged(Sensor sensor, int accuracy)
  {
    // TODO Auto-generated method stub
  }

  @Override
  public void onSensorChanged(SensorEvent event)
  {
    switch(event.sensor.getType())
    {
    case Sensor.TYPE_ACCELEROMETER:
      acc = event.values;
      break;
    case Sensor.TYPE_GYROSCOPE:
      gyro = event.values;
      break;
    case Sensor.TYPE_AMBIENT_TEMPERATURE:
      temp = event.values;
      break;
    case Sensor.TYPE_PRESSURE:
      pressure = event.values;
      break;
    case Sensor.TYPE_GRAVITY:
      gravity = event.values;
      break;
    case Sensor.TYPE_LIGHT:
      light = event.values;
      break;
    case Sensor.TYPE_MAGNETIC_FIELD:
      mfield = event.values;
      break;
    case Sensor.TYPE_RELATIVE_HUMIDITY:
      rh = event.values;
      break;
    case Sensor.TYPE_ROTATION_VECTOR:
      rotation = event.values;
      break;
    case Sensor.TYPE_PROXIMITY:
      proximity = event.values;
      break;
    }
  }

  public String readSensorValues(String sensorType)
  {
    if(sensorType == null || sensorType.equals(""))
    {
      return null;
    }

    if(sensorType.equals("acc"))
    {
      return java.util.Arrays.toString(acc);
    }
    else if(sensorType.equals("gyro"))
    {
      return java.util.Arrays.toString(gyro);
    }
    else if(sensorType.equals("temp"))
    {
      return java.util.Arrays.toString(temp);
    }
    else if(sensorType.equals("pressure"))
    {
      return java.util.Arrays.toString(pressure);
    }
    else if(sensorType.equals("light"))
    {
      return java.util.Arrays.toString(light);
    }
    else if(sensorType.equals("mfield"))
    {
      return java.util.Arrays.toString(mfield);
    }
    else if(sensorType.equals("rh"))
    {
      return java.util.Arrays.toString(rh);
    }
    else if(sensorType.equals("rotation"))
    {
      return java.util.Arrays.toString(rotation);
    }
    else if(sensorType.equals("proximity"))
    {
      return java.util.Arrays.toString(proximity);
    }
    return null;
  }
}
```

Listing 8: InternalSensorListener.java

```
package com.nurgak.trebla;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.ScrollView;
import android.widget.TextView;

```

```java
public class MainActivity extends Activity
{
  ScrollView logScroller;
  TextView logTextView;
  Button buttonUpdateLog, buttonStopApplication;

  String line;
  String separator;
  Process mProcess;
  BufferedReader reader;
  StringBuilder builder = new StringBuilder();

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);

    // set activity layout
    setContentView(R.layout.activity_main);

    logScroller = (ScrollView) findViewById(R.id.logScroller);

    logTextView = (TextView) findViewById(R.id.logTextView);

    buttonUpdateLog = (Button) findViewById(R.id.buttonUpdateLog);
    buttonUpdateLog.setOnClickListener(updateLog);

    buttonStopApplication = (Button) findViewById(R.id.buttonStopApplication);
    buttonStopApplication.setOnClickListener(stopApplication);

    separator = System.getProperty("line.separator");
  }

  View.OnClickListener updateLog = new View.OnClickListener()
  {
    public void onClick(View v)
    {
      try
      {
        mProcess = Runtime.getRuntime().exec("logcat -d -t 30 trebla:v *:s");
        reader = new BufferedReader(new InputStreamReader(mProcess.getInputStream()));
        builder.delete(0, builder.length());

        while((line = reader.readLine()) != null)
        {
          builder.append(line);
          builder.append(separator);
        }
        logTextView.setText(builder.toString());
        logScroller.fullScroll(View.FOCUS_DOWN);
      }
      catch(IOException e)
      {
      }
    }
  };

  View.OnClickListener stopApplication = new View.OnClickListener()
  {
    public void onClick(View v)
    {
      finish();
      System.exit(0);
    }
  };

  @Override
  public boolean onCreateOptionsMenu(Menu menu)
  {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
  }
}
```

Listing 9: MainActivity.java

```java
package com.nurgak.trebla.services;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.NetworkInterface;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.util.Enumeration;
import org.apache.http.conn.util.InetAddressUtils;

import com.nurgak.trebla.BoundService;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.util.Log;

public class SocketServerService extends BoundService implements Runnable
{
  ServerSocket serverSocket = null;
  Socket clientSocket = null;
  PrintWriter output = null;

  static InputStreamReader inputStreamReader = null;
  static BufferedReader bufferedReader = null;
  static String msg = null;

  SocketServerReceiver socketServerReceiver = null;

  int serverPort = 8888;

  public SocketServerService()
  {
    // start the socket server and listen to incoming client requests
    try
    {
      serverSocket = new ServerSocket(serverPort);
    }
    catch(IOException e)
    {
      e.printStackTrace();
    }

    Log.d("trebla", "IP: " + getLocalIpAddress() + ":" + serverPort);

    // check if the socket is open to start the listening thread
    new Thread(this).start();
  }

  private void socketServerStop()
  {
    output.close();
    output = null;

    try
    {
      bufferedReader.close();
      bufferedReader = null;
      inputStreamReader.close();
      inputStreamReader = null;
      clientSocket.close();
      clientSocket = null;
    }
    catch(IOException e)
    {
      e.printStackTrace();
    }
  }

  @Override
  public void run()
  {
    socketServerReceiver = new SocketServerReceiver();

    // start the broadcast receiver
    IntentFilter filter = new IntentFilter();
    filter.addAction("trebla");

    // TODO this throws a null pointer exception sometimes for some reason, maybe the receiver is not ready, try to loop while its null
    while(socketServerReceiver == null);
    registerReceiver(socketServerReceiver, filter);

    while(true)
    {
      Log.d("trebla", "Waiting for client");
      try
      {
        // accept the client connection (blocking)
        clientSocket = serverSocket.accept();

        inputStreamReader = new InputStreamReader(clientSocket.getInputStream());
        bufferedReader = new BufferedReader(inputStreamReader);
        output = new PrintWriter(clientSocket.getOutputStream(), true);
      }
      catch(IOException e)
      {
        e.printStackTrace();
        break;
      }

      bindTreblaService();

      // get the client message
      while(true)
      {
        Log.d("trebla", "Listening");
        // read from client (blocking)
        try
        {
          msg = bufferedReader.readLine();
        }
        catch(IOException e)
        {
          e.printStackTrace();
        }

        // if not disconnecting on msg == null it loops indefinitely for some reason
        if(msg == null || msg.length() == 0 || msg.equals("close") || msg.equals("exit") || msg.equals("quit"))
        {
          Log.d("trebla", "Disconnecting");
          break;
        }

        Log.d("trebla", "Input: '" + msg + "'");

        // actually use the bounded trebla service
        if(treblaService == null)
        {
          // user might have connected and disconnected too fast and the instance might be null
          break;
        }

        output.append(treblaService.processMessage(msg));

        output.append("\n");
        output.flush();
      }

      unBindTreblaService();
      socketServerStop();
    }
  }

  @Override
  public void onDestroy()
  {
    // when something goes wrong inform the client
    if(output != null)
    {
      output.write("closing");
      output.flush();
      socketServerStop();
    }
    super.onDestroy();
  }

  // this function gets the local IPv4 address to show the user
  private String getLocalIpAddress()
  {
    try
    {
      for(Enumeration<NetworkInterface> en = NetworkInterface.getNetworkInterfaces(); en.
          hasMoreElements();)
      {
        NetworkInterface intf = en.nextElement();
        for(Enumeration<InetAddress> enumIpAddr = intf.getInetAddresses(); enumIpAddr.
            hasMoreElements();)
        {
          InetAddress inetAddress = enumIpAddr.nextElement();
          if(!inetAddress.isLoopbackAddress() && InetAddressUtils.isIPv4Address(inetAddress.
              getHostAddress()))
          {
            return inetAddress.getHostAddress().toString();
          }
        }
      }
    }
    catch(SocketException e)
    {
      e.printStackTrace();
    }
    return "No IP Available";
  }

  public class SocketServerReceiver extends BroadcastReceiver
  {
    @Override
    public void onReceive(Context context, Intent intent)
    {
      // if a client is connected forward the data to it
      if(output != null)
```

```
      {
        output.append(intent.getStringExtra("data") + "\n");
        output.flush();
      }
    }
  }
}
```

Listing 10: SocketServerService.java

```
package com.nurgak.trebla;

import com.nurgak.trebla.services.SocketServerService;

import android.app.Application;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.IBinder;

public class TreblaApplication extends Application
{
  BoundService commService;
  ServiceConnection serviceConnection;

  @Override
  public void onCreate()
  {
    // bind the connection service to the application
    // here the service is the socket connection service by default, when other options become available this shall be changed
    Intent intent = new Intent(this, SocketServerService.class);
    //Intent intent = new Intent(this, BluetoothClientService.class);
    serviceConnection = new ServiceConnection()
    {
      @Override
      public void onServiceConnected(ComponentName name, IBinder binder)
      {
        // bounded service instance which can be used to call its methods directly
        commService = ((BoundService.LocalBinder) binder).getBoundService();
      }

      @Override
      public void onServiceDisconnected(ComponentName name)
      {
        // TODO Auto-generated method stub
      }
    };

    if(bindService(intent, serviceConnection, Context.BIND_AUTO_CREATE))
    {
      // communication service is bound and up
    }

    super.onCreate();
  }

  @Override
  public void onTerminate()
  {
    // when the activity is stopped the communication service must be unbound
    unbindService(serviceConnection);

    super.onTerminate();
  }
}
```

Listing 11: TreblaApplication.java

```
package com.nurgak.trebla;

import android.util.Log;

import com.nurgak.trebla.services.BatteryBroadcastReceiver;
import com.nurgak.trebla.services.Command;
import com.nurgak.trebla.services.InternalSensorListener;
import com.nurgak.trebla.services.UsbCommunicationManager;

public class TreblaService extends BoundService
{
  // command parser
  Command cmd = null;

  // internal sensors
  InternalSensorListener sensorListener = null;

  // battery
  BatteryBroadcastReceiver batteryReceiver = null;

  // usb
  UsbCommunicationManager usb = null;

  // bluetooth
  // TODO

  // camera
  // TODO

  // gps
  // TODO

  String returnValue = "";

  public String processMessage(String msg)
  {
    // parse the command
    cmd = new Command(msg);

    returnValue = "";

    if(cmd.command.equals("sensor"))
    {
      if(cmd.argument.equals("start"))
      {
        // create and start the internal sensor listener
        sensorListener = new InternalSensorListener(this);
        sensorListener.start();
        returnValue += "internal sensor listener started";
      }
      else if(cmd.argument.equals("stop") && sensorListener != null)
      {
        // stop and remove all references to the sensor listener
        sensorListener.stop();
        sensorListener = null;
        returnValue += "internal sensor listener stopped";
      }
      else if(!cmd.argument.equals("") && sensorListener != null)
      {
        returnValue += sensorListener.readSensorValues(cmd.argument);
      }
      else if(cmd.argument.equals("") && sensorListener != null)
      {
        returnValue += "no sensor type specified";
      }
      else if(sensorListener == null)
      {
        returnValue += "internal sensor listener not started";
      }
    }
    else if(cmd.command.equals("battery"))
    {
      if(cmd.argument.equals("start") && batteryReceiver == null)
      {
        // start battery state listener
        batteryReceiver = new BatteryBroadcastReceiver(this);
        batteryReceiver.start();
        returnValue += "battery listener started";
      }
      else if(cmd.argument.equals("stop") && batteryReceiver != null)
      {
        // stop listening to battery state
        batteryReceiver.stop();
        batteryReceiver = null;
        returnValue += "battery listener stopped";
      }
      else if(cmd.argument.equals("state") && batteryReceiver != null)
      {
        returnValue += batteryReceiver.getBatteryState();
      }
      else if(!cmd.argument.equals("") && batteryReceiver != null)
      {
        returnValue += "no argument specified";
      }
      else if(batteryReceiver == null)
      {
        returnValue += "battery listener not started";
      }
    }
    else if(cmd.command.equals("usb"))
    {
      if(usb == null)
      {
        usb = new UsbCommunicationManager(this);
      }

      if(cmd.checkFlag("l") || cmd.checkFlag("list"))
      {
        returnValue += usb.listUsbDevices();
      }
      else if(cmd.checkFlag("r") || cmd.checkFlag("read"))
      {
        StringBuilder data = new StringBuilder();
        usb.read(data);
        returnValue += data.toString();
      }
      else if((cmd.checkFlag("w") || cmd.checkFlag("write")) && !cmd.argument.equals(""))
      {
        returnValue += "Transferred bytes: " + usb.write(cmd.argument);
      }
      else if(cmd.argument.equals("connect"))
      {
        usb.connect();
        returnValue += "Trying to establish connection";
      }
    }
    else if(cmd.command.equals("picture") || cmd.command.equals("image") || cmd.command.equals("photo"))
    {
      returnValue += "not implemented yet";
    }
    else if(cmd.command.equals("gps"))
    {
      returnValue += "not implemented yet";
    }
    else
    {
      returnValue += "unrecognised command";
    }
    return returnValue;
  }

  @Override
  public void onDestroy()
  {
    Log.d("trebla", "Stopping all bounded services");

    // make sure all services are shut down
    if(usb != null)
    {
      usb.stop();
      usb = null;
    }

    if(sensorListener != null)
    {
      sensorListener.stop();
      sensorListener = null;
    }

    if(batteryReceiver != null)
    {
      batteryReceiver.stop();
      batteryReceiver = null;
    }
    super.onDestroy();
  }
}
```

Listing 12: TreblaService.java

```
package com.nurgak.trebla.services;

import java.nio.ByteBuffer;
import java.util.HashMap;
import java.util.Iterator;

import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.hardware.usb.UsbConstants;
import android.hardware.usb.UsbDevice;
import android.hardware.usb.UsbDeviceConnection;
import android.hardware.usb.UsbEndpoint;
import android.hardware.usb.UsbInterface;
import android.hardware.usb.UsbManager;
import android.hardware.usb.UsbRequest;
import android.util.Log;

public class UsbCommunicationManager implements Runnable
{
  static final String ACTION_USB_PERMISSION = "com.android.example.USB_PERMISSION";

  UsbManager usbManager;
  UsbDevice usbDevice = null;
  UsbInterface usbCdcInterface = null;
  UsbInterface usbHidInterface = null;
  UsbEndpoint usbCdcRead = null;
  UsbEndpoint usbCdcWrite = null;
  UsbDeviceConnection usbCdcConnection;

  Thread readThread = null;
  volatile boolean readThreadRunning = true;

  PendingIntent permissionIntent;

```

```java
  Context context;

  byte[] readBytes = new byte[256];

  public UsbCommunicationManager(Context context)
  {
    this.context = context;
    usbManager = (UsbManager) context.getSystemService(Context.USB_SERVICE);

    // ask permission from user to use the usb device
    permissionIntent = PendingIntent.getBroadcast(context, 0, new Intent(ACTION_USB_PERMISSION), 0);
    IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
    context.registerReceiver(usbReceiver, filter);
  }

  public void connect()
  {
    // check if there's a connected usb device
    if(usbManager.getDeviceList().isEmpty())
    {
      Log.d("trebla", "No connected devices");
      return;
    }

    // get the first (only) connected device
    usbDevice = usbManager.getDeviceList().values().iterator().next();

    // user must approve of connection if not in the /res/usb_device_filter.xml file
    usbManager.requestPermission(usbDevice, permissionIntent);
  }

  public void stop()
  {
    usbDevice = null;
    usbCdcInterface = null;
    usbHidInterface = null;
    usbCdcRead = null;
    usbCdcWrite = null;

    context.unregisterReceiver(usbReceiver);
  }

  public String write(String data)
  {
    if(usbDevice == null)
    {
      return "no usb device selected";
    }

    int sentBytes = 0;
    if(!data.equals(""))
    {
      synchronized(this)
      {
        // send data to usb device
        byte[] bytes = data.getBytes();
        sentBytes = usbCdcConnection.bulkTransfer(usbCdcWrite, bytes, bytes.length, 1000);
      }
    }

    return Integer.toString(sentBytes);
  }

  public String read(StringBuilder dest)
  {
    if(usbCdcRead == null)
    {
      return "not connected to a device";
    }

    String state = "";

    if(readThread != null && readThread.isAlive())
    {
      readThreadRunning = false;
      state = "stopping usb listening thread";
    }
    else
    {
      readThreadRunning = true;
      readThread = new Thread(this);
      readThread.start();
      state = "starting usb listening thread";
    }

    return state;

    //    if(usbDevice == null)
    //    {
    //      return "no usb device selected";
    //    }
    //
    //    // reinitialize read value byte array
    //    //Arrays.fill(readBytes, (byte) 0);
    //
    //    // wait for some data from the mcu
    //    int recvBytes = usbCdcConnection.bulkTransfer(usbCdcRead, readBytes, readBytes.length, 3000);
    //
    //    if(recvBytes > 0)
    //    {
    //      for(int i = 0; i < recvBytes; ++i)
    //      {
    //        dest.append((char) readBytes[i]);
    //      }
    //
    //      Log.d("trebla", "Got some data: " + dest.toString());
    //    }
    //    else
    //    {
    //      Log.d("trebla", "Did not get any data: " + recvBytes);
    //    }
    //
    //    return Integer.toString(recvBytes);
  }

  public String listUsbDevices()
  {
    HashMap<String, UsbDevice> deviceList = usbManager.getDeviceList();

    if(deviceList.size() == 0)
    {
      return "no usb devices found";
    }

    Iterator<UsbDevice> deviceIterator = deviceList.values().iterator();
    String returnValue = "";
    UsbInterface usbInterface;

    while(deviceIterator.hasNext())
    {
      UsbDevice device = deviceIterator.next();
      returnValue += "Name: " + device.getDeviceName();
      returnValue += "\nID: " + device.getDeviceId();
      returnValue += "\nProtocol: " + device.getDeviceProtocol();
      returnValue += "\nClass: " + device.getDeviceClass();
      returnValue += "\nSubclass: " + device.getDeviceSubclass();
      returnValue += "\nProduct ID: " + device.getProductId();
      returnValue += "\nVendor ID: " + device.getVendorId();
      returnValue += "\nInterface count: " + device.getInterfaceCount();

      for(int i = 0; i < device.getInterfaceCount(); i++)
      {
        usbInterface = device.getInterface(i);
        returnValue += "\n  Interface " + i;
        returnValue += "\n\tInterface ID: " + usbInterface.getId();
        returnValue += "\n\tClass: " + usbInterface.getInterfaceClass();
        returnValue += "\n\tProtocol: " + usbInterface.getInterfaceProtocol();
        returnValue += "\n\tSubclass: " + usbInterface.getInterfaceSubclass();
        returnValue += "\n\tEndpoint count: " + usbInterface.getEndpointCount();

        for(int j = 0; j < usbInterface.getEndpointCount(); j++)
        {
          returnValue += "\n\t  Endpoint " + j;
          returnValue += "\n\t\tAddress: " + usbInterface.getEndpoint(j).getAddress();
          returnValue += "\n\t\tAttributes: " + usbInterface.getEndpoint(j).getAttributes();
          returnValue += "\n\t\tDirection: " + usbInterface.getEndpoint(j).getDirection();
          returnValue += "\n\t\tNumber: " + usbInterface.getEndpoint(j).getEndpointNumber();
          returnValue += "\n\t\tInterval: " + usbInterface.getEndpoint(j).getInterval();
          returnValue += "\n\t\tType: " + usbInterface.getEndpoint(j).getType();
          returnValue += "\n\t\tMax packet size: " + usbInterface.getEndpoint(j).
              getMaxPacketSize();
        }
      }
    }

    return returnValue;
  }

  private void setupConnection()
  {
    // find the right interface
    for(int i = 0; i < usbDevice.getInterfaceCount(); i++)
    {
      // communications device class (CDC) type device
      if(usbDevice.getInterface(i).getInterfaceClass() == UsbConstants.USB_CLASS_CDC_DATA)
      {
        usbCdcInterface = usbDevice.getInterface(i);

        // find the endpoints
        for(int j = 0; j < usbCdcInterface.getEndpointCount(); j++)
        {
          if(usbCdcInterface.getEndpoint(j).getType() == UsbConstants.USB_ENDPOINT_XFER_BULK)
          {
            if(usbCdcInterface.getEndpoint(j).getDirection() == UsbConstants.USB_DIR_OUT)
            {
              // from host to device
              usbCdcWrite = usbCdcInterface.getEndpoint(j);
            }

            if(usbCdcInterface.getEndpoint(j).getDirection() == UsbConstants.USB_DIR_IN)
            {
              // from device to host
              usbCdcRead = usbCdcInterface.getEndpoint(j);
            }
          }
        }
      }
    }
  }

  private final BroadcastReceiver usbReceiver = new BroadcastReceiver()
  {
    public void onReceive(Context context, Intent intent)
    {
      String action = intent.getAction();
      if(ACTION_USB_PERMISSION.equals(action))
      {
        // broadcast is like an interrupt and works asynchronously with the class, it must be synced just in case
        synchronized(this)
        {
          if(intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false))
          {
            // fetch all the endpoints
            setupConnection();

            // open and claim interface
            usbCdcConnection = usbManager.openDevice(usbDevice);
            usbCdcConnection.claimInterface(usbCdcInterface, true);

            // set dtr to true (ready to accept data)
            usbCdcConnection.controlTransfer(0x21, 0x22, 0x1, 0, null, 0, 0);

            // set flow control to 8N1 at 9600 baud
            /* int baudRate = 9600; byte stopBitsByte = 1; byte
             * parityBitesByte = 0; byte dataBits = 8; byte[] msg =
             * { (byte) (baudRate & 0xff), (byte) ((baudRate >> 8) &
             * 0xff), (byte) ((baudRate >> 16) & 0xff), (byte)
             * ((baudRate >> 24) & 0xff), stopBitsByte,
             * parityBitesByte, (byte) dataBits }; */

            //Log.d("trebla", "Flow: " + connection.controlTransfer(0x21, 0x20, 0, 0, new byte[] {(byte) 0x80, 0x25, 0x00, 0x00, 0x00, 0x00, 0x08}, 7, 0));

            //connection.controlTransfer(0x21, 0x20, 0, 0, msg, msg.length, 5000);
          }
          else
          {
            Log.d("trebla", "Permission denied for USB device");
          }
        }
      }
      else if(UsbManager.ACTION_USB_DEVICE_DETACHED.equals(action))
      {
        if(usbDevice != null)
        {
          usbCdcConnection.releaseInterface(usbCdcInterface);
          usbCdcConnection.close();
          usbCdcConnection = null;
          usbDevice = null;
          Log.d("trebla", "USB connection closed");
        }
      }
    }
  };

  @Override
  public void run()
  {
    Log.d("trebla", "Started the usb linstener");
    ByteBuffer buffer = ByteBuffer.allocate(255);
    UsbRequest request = new UsbRequest();
    request.initialize(usbCdcConnection, usbCdcRead);

    String dataByte, data = "";
    int packetState = 0;

    while(readThreadRunning)
    {
      // queue a request on the interrupt endpoint
      request.queue(buffer, buffer.capacity());
      // wait for status event
```

```java
      if(usbCdcConnection.requestWait() == request)
      {
        // there is no way to know how many bytes are coming, so simply forward the non-null values

        for(int i = 0; i < buffer.capacity() && buffer.get(i) != 0 ; i++)
        {
          // transform ascii (0-255) to its character equivalent and append
          dataByte = Character.toString((char) buffer.get(i));
          if(packetState == 0 && dataByte.equals("["))
          {
            // start
            packetState = 1;
            data += dataByte;
          }
          else if(packetState == 1 && !dataByte.equals("]"))
          {
            // in-between
            data += dataByte;
          }
          else if(packetState == 1 && dataByte.equals("]"))
          {
            // end
            packetState = 2;
            data += dataByte;
            break;
          }
        }

        if(packetState == 2)
        {
          // send data to client
          Intent intent = new Intent();
          intent.setAction("trebla");
          intent.putExtra("data", data);
          context.sendBroadcast(intent);

          // reset packet
          packetState = 0;
          data = "";
        }
      }
      else
      {
        Log.e("trebla", "Was not able to read from USB device, ending listening thread");
        readThreadRunning = false;
        break;
      }
    }
  }
}
```

Listing 13: UsbCommunicationManager.java

A.3 Interface

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/action_settings"/>

</menu>
```

Listing 14: main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <ScrollView
        android:id="@+id/logScroller"
        android:layout_width="match_parent"
        android:layout_height="0dip"
        android:layout_weight="1" >

        <TextView
            android:id="@+id/logTextView"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
    </ScrollView>

    <LinearLayout
        style="?android:attr/buttonBarStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <Button
            android:id="@+id/buttonUpdateLog"
            style="?android:attr/buttonBarButtonStyle"
            android:layout_width="0dip"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/updatelog" />

        <Button
            android:id="@+id/buttonStopApplication"
            style="?android:attr/buttonBarButtonStyle"
            android:layout_width="0dip"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/stopapplication" />
    </LinearLayout>

```

```
</LinearLayout>
```

Listing 15: activity_main.xml

A.4 Resources

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Trebla</string>
    <string name="action_settings">Settings</string>
    <string name="updatelog">Update log</string>
    <string name="stopapplication">Stop application</string>

</resources>
```

Listing 16: strings.xml

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <usb-device vendor-id="9025" product-id="32822" />
</resources>
```

Listing 17: usb_device_filter.xml

B Arduino source code

```
#include <Wire.h>

#define CTRL_REG1 0x20
#define CTRL_REG2 0x21
#define CTRL_REG3 0x22
#define CTRL_REG4 0x23
#define CTRL_REG5 0x24

char sprintBuffer[64];

byte L3G4200D_address = 0b1101000;

int x, y, z;

byte i2c_address, i2c_register, i2c_value;

void setup()
{
  // start i2c, automatically enables pullups to 5V
  Wire.begin();
  // turn off internal i2c pullups, external pullups to 3.3V are on the sensor
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  Serial.begin(9600);
  // configure L3G4200  - 250, 500 or 2000 deg/sec
  setupL3G4200D(250);
  // wait for the sensor to be ready
  delay(1500);
}

void loop()
{
  if(Serial.available())
  {
    switch(Serial.read())
    {
    case ’s’:
      // configure L3G4200
      setupL3G4200D(2000);
      break;
    case ’g’:
      // update x, y, and z with new values
      getGyroValues();
      // sprintf does not support float values, so send raw values
      sprintf(sprintBuffer, "[%d,%d,%d]", x, y, z);
      Serial.print(sprintBuffer);
      break;
    case ’r’:
      // read from i2c
      i2c_address = Serial.read();
      i2c_register = Serial.read();
      Serial.write(readRegister(i2c_address, i2c_register));
      break;
    case ’w’:
      // write to i2c
      i2c_address = Serial.read();
      i2c_register = Serial.read();
      i2c_value = Serial.read();
      writeRegister(i2c_address, i2c_register, i2c_value);
      break;
    }
  }
}

void getGyroValues()
{
  byte xMSB = readRegister(L3G4200D_address, 0x29);
  byte xLSB = readRegister(L3G4200D_address, 0x28);
  x = ((xMSB << 8) | xLSB);

  byte yMSB = readRegister(L3G4200D_address, 0x2B);
  byte yLSB = readRegister(L3G4200D_address, 0x2A);
  y = ((yMSB << 8) | yLSB);

  byte zMSB = readRegister(L3G4200D_address, 0x2D);
  byte zLSB = readRegister(L3G4200D_address, 0x2C);
  z = ((zMSB << 8) | zLSB);
}

```

```
int setupL3G4200D(int scale)
{
  // From  Jim Lindblom of Sparkfun's code

  // Enable x, y, z and turn off power down:
  writeRegister(L3G4200D_address, CTRL_REG1, 0b00001111);

  // If you'd like to adjust/use the HPF, you can edit the line below to configure CTRL_REG2:
  writeRegister(L3G4200D_address, CTRL_REG2, 0b00000000);

  // Configure CTRL_REG3 to generate data ready interrupt on INT2
  // No interrupts used on INT1, if you'd like to configure INT1 or INT2 otherwise, consult the datasheet:
  //writeRegister(L3G4200D_address, CTRL_REG3, 0b00001000);

  // CTRL_REG4 controls the full-scale range, among other things:

  if(scale == 250)
  {
    writeRegister(L3G4200D_address, CTRL_REG4, 0b00000000);
  }
  else if(scale == 500)
  {
    writeRegister(L3G4200D_address, CTRL_REG4, 0b00010000);
  }
  else
  {
    writeRegister(L3G4200D_address, CTRL_REG4, 0b00110000);
  }

  // CTRL_REG5 controls high-pass filtering of outputs, use it
  // if you'd like:
  writeRegister(L3G4200D_address, CTRL_REG5, 0b00000000);
}

void writeRegister(byte address, byte registerAddress, byte val)
{
  Wire.beginTransmission(address);
  Wire.write(registerAddress);
  Wire.write(val);
  Wire.endTransmission();
}

int readRegister(int address, byte registerAddress)
{
  Wire.beginTransmission(address);
  Wire.write(registerAddress);
  Wire.endTransmission();

  Wire.requestFrom(address, 1);

  // waiting
  while(!Wire.available());
  return Wire.read();
}
```

Listing 18: Python source code

C Accelerometer sampling Matlab source code

```
%ip = '192.168.11.2';
ip = '192.168.0.105';
%ip = '128.179.166.25';
port = 8888;
run = true;

close all

t = tcpip(ip, port, 'NetworkRole', 'client');

% these delays are necessary for some reason
try
    fopen(t);
    pause(.1);
    fprintf(t, 'sensor start');
    pause(.1);
catch exception
    disp('Could not connect, did you launch the application?');
    return
end

figure('name','Internal accelerometer real-time view');

% add a stop button to the plot
buttonStop = uicontrol('Style', 'pushbutton', 'Callback', 'run = false;', 'String', 'Stop');

% real returned values
subplot(1,2,1);

h = bar([0 0 0],'r');
title('BOSCH BMA250');

% label axes and set limits
xlabel('Axis');
ylabel('Acceleration (m/s^2)');
set(gca,'XTickLabel',{'x', 'y', 'z'})
set(gca,'YLim',[-10 10]);

% 3d image of phone position
subplot(1,2,2);

title('Smartphone model');

% make the elongated box centered on the origin
my_vertices = [
    -1 -.5 -.12
    -1 .5 -.12
    1 .5 -.12
    1 -.5 -.12
    -1 -.5 .12
    -1 .5 .12
    1 .5 .12
    1 -.5 .12];

my_faces = [
    1 2 3 4
    2 6 7 3
    4 3 7 8
    1 5 8 4
    1 2 6 5
    5 6 7 8];

% make a 3D plot
view(3)
% set default rotation
view([90 0 0])
% add some perspective to make it look nice
camproj(’perspective’)

% force same length on all axis
axis vis3d
axis([-1 1 -1 1 -1 1 0 1])

% hide axis
axis off;

% make the actual 3D model
d = patch(’Vertices’, my_vertices, ’Faces’, my_faces, ’FaceColor’, ’b’);

% original values used later to reset model rotation
reset = [get(d, ’XData’); get(d, ’YData’); get(d, ’ZData’)];

% start sampling
fprintf(t, ’sensor acc’);
pause(.1);

% as long as the connection is up loop here
while run
    if get(t, ’BytesAvailable’) > 1
        % read data in, make the right format
        data = str2num(char(fread(t, t.BytesAvailable, ’char’)’));
        % update plot
        if length(data) == 3
            try
                set(h, ’YData’, data);
            catch error
                continue
            end

            % get new rotation angles for the 3D model
            x = data(1) / 9.8;
            y = data(2) / 9.8;
            angleX = asin(x)*180/pi;
            angleY = asin(y)*180/pi;

            if data(3) < 0
                angleX = angleX + 180;
                angleY = -angleY;
            end

            % reset box to initial view
            set(d, ’Xdata’, reset(1:4,1:6));
            set(d, ’Ydata’, reset(5:8,1:6));
            set(d, ’Zdata’, reset(9:12,1:6));

            % rotate the model around the origin
            rotate(d, [1,0,0], angleX, [0 0 0]);
            rotate(d, [0,1,0], angleY, [0 0 0]);

            % force redraw
            drawnow
        end
        % ask for another data point
        fprintf(t, ’sensor acc’);
    end
end

fclose(t);
delete(t);
clear t

close all
```

Listing 19: Matlab source code for accelerometer sampling

D Gyroscope sampling Matlab source code

```
ip = ’192.168.0.107’;
%ip = ’128.179.164.69’;
port = 8888;
run = true;

xLimit = 100;
sampledValues = zeros(xLimit,2);

yLimit = 200;

% allow some reading retries when waiting for incoming data
retries = 10;

close all

t = tcpip(ip, port, ’NetworkRole’, ’client’);

% these delays are necessary for some reason
```

```
try
    fopen(t);
    pause(.1);
    fprintf(t, 'sensor start');
    pause(.1);
    % start the usb communication
    fprintf(t, 'usb connect');
    pause(.1);
    % start reading what comes from the usb port
    fprintf(t, 'usb -r');
    pause(.1);
catch exception
    disp('Could not connect, did you launch the application?');
    return
end

figure('name','Internal and external gyroscope real-time view');

subplot(2,2,1);

% add a stop button to the plot
buttonStop = uicontrol('Style', 'pushbutton', 'Callback', 'run = false;', 'String', 'Stop');

int = bar([0 0 0],'r');
title('Internal gyroscope (InvenSense MPU3050)');

% label axes and set limits
xlabel('Axis');
ylabel('Rotation (0/s)');
set(gca,'XTickLabel',{'x', 'y', 'z'})
set(gca,'YLim',[-yLimit yLimit]);

subplot(2,2,2);

ext = bar([0 0 0],'b');
title('External gyroscope (STMicroelectronics L3G4200D)');

% label axes and set limits
xlabel('Axis');
ylabel('Rotation (0/s)');
set(gca,'XTickLabel',{'x', 'y', 'z'})
set(gca,'YLim',[-yLimit yLimit]);

subplot(2,2,3:4);

% plot the values over time
tim = plot([1:length(sampledValues)], sampledValues,'YDataSource','sampledValues');
set(gcf, 'DefaultAxesColorOrder', [1 0 0;0 0 1]);
title('Time series: gyroscope values');

set(tim(1),'Displayname','Internal');
set(tim(2),'Displayname','External');
legend('Location','north');

xlabel('Time');
```

```
ylabel(’Rotation (0/s)’);
%set(gca,’YLim’,[-yLimit yLimit]);

% start sampling
fprintf(t, ’sensor gyro’);
pause(.1);

% 1 = internal, 2 = external
sensor = 1;
retry = retries;
timeout = clock;

rad2deg = 180 / pi;

timeSeriesStart = clock;

% as long as the connection is up loop here
while run
    if get(t, ’BytesAvailable’) > 1
        % read data in, make the right format
        raw = fread(t, t.BytesAvailable, ’char’)’;
        data = str2num(char(raw(find(raw == ’[’, 1):length(raw)))) * rad2deg;

        % update plot
        if length(data) == 3
            try
                if sensor == 1
                    % do not allow more then xLimit values for the time series
                    if length(sampledValues) > xLimit - 1
                        sampledValues = sampledValues(2:end,:);
                    end
                    sampledValues = [sampledValues; data(3) 0];
                    set(int, ’YData’, data);
                else
                    data = data / (36 / 0.0058) + 0.5;
                    sampledValues(end, 2) = data(3);

                    refreshdata(tim, ’caller’);
                    set(ext, ’YData’, data);
                    refreshdata
                    drawnow
                end
                retry = 0;
            catch error
                continue
            end
        elseif retry > 0
            retry = retry - 1;
        end

        % ask for another data point, alternate sensors
        if retry == 0
            if sensor == 1
                sensor = 2;
                fprintf(t, ’usb -w g’);
            else
                sensor = 1;
                fprintf(t, ’sensor gyro’);
            end
            timeout = clock;
            retry = retries;
        end
    end

    % sometimes this hangs, quit gracefully
    if etime(clock, timeout) > 3
        disp(’Connection lost.’);
        break
    end
end

fclose(t);
delete(t);
clear t

close all
```

Listing 20: Matlab source code for accelerometer sampling

E Python terminal source code

```
#!/usr/bin/env python2.7
import socket
import time
import threading
import wx
import select

connectionState = 0
dataBuffer = ""
sentCommandHistory = []
sentCommandHistoryId = 0

EVENT_DISCONNECTED = wx.NewEventType()
EVENT_NEWDATA = wx.NewEventType()

class SocketClientThread(threading.Thread):
    def __init__(self, conn, parent):
        threading.Thread.__init__(self)
        self.conn = conn
        self.parent = parent

    def run(self):
        global connectionState, dataBuffer
        while connectionState:
            # this will try to recieve data continuously
            try:
                # this makes sure there’s someting to read from the server before actually reading it
                # it’s necessary because without it will keep blocking even with a closed connection
                if select.select([self.conn], [], []):
                    # this is blocking
                    data = self.conn.recv(1024)

                # connection lost
                if not data:
                    wx.PostEvent(self.parent, wx.PyCommandEvent(EVENT_DISCONNECTED, -1))
                    return

                # remove all whitespace characters on the right side
                # append so that if there’s a new packet before the previous is shown it doesn’t get lost
                dataBuffer += data.rstrip()
                wx.PostEvent(self.parent, wx.PyCommandEvent(EVENT_NEWDATA, -1))
            except socket.timeout, e:
                #print e.args[0]
                continue
            except Exception as e:
                #print e.args[0]
                return

    def send(self, data):
        # new line defines an end of command on the server side
        self.conn.send(data + "\n")

    def close(self):
        global connectionState
        connectionState = 0
        self.conn.close()

class SocketClientUI(wx.Frame):
    def __init__(self, parent, title):
        super(SocketClientUI, self).__init__(parent, title=title, size=(500, 500))

        # keyboard events
        self.Bind(wx.EVT_CHAR_HOOK, self.onKey)

        # bind events from socket client thread
        self.Bind(wx.PyEventBinder(EVENT_DISCONNECTED, 1), self.disconnect)
        self.Bind(wx.PyEventBinder(EVENT_NEWDATA, 1), self.updateOutput)

        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):
        panel = wx.Panel(self)
        sizer = wx.GridBagSizer(5, 5)

        # add some labels for IP and port text entries
        sizer.Add(wx.StaticText(panel, label="IP"), pos=(0, 0), flag=wx.TOP|wx.LEFT, border=5)
        sizer.Add(wx.StaticText(panel, label="Port"), pos=(0, 3), flag=wx.TOP, border=5)

        # set a default value for the IP
        self.tc_ip = wx.TextCtrl(panel, value="192.168.0.105")
        sizer.Add(self.tc_ip, pos=(1, 0), span=(1, 3), flag=wx.EXPAND|wx.LEFT, border=5)

        # port is 8888 by defaut because of reasons
        self.tc_port = wx.TextCtrl(panel, value="8888")
        sizer.Add(self.tc_port, pos=(1, 3), span=(1, 1), flag=wx.EXPAND)

        # click connect button for magic
        self.button_connect = wx.Button(panel, label="Connect")
        sizer.Add(self.button_connect, pos=(1, 4), flag=wx.EXPAND|wx.RIGHT, border=5)
        self.button_connect.Bind(wx.EVT_BUTTON, self.connect)

        # output data
        sizer.Add(wx.StaticText(panel, label="Output"), pos=(2, 0), flag=wx.TOP|wx.LEFT, border=5)
        self.output = wx.TextCtrl(panel, style = wx.TE_MULTILINE)
        #self.output.Enable(False)
        sizer.Add(self.output, pos=(3, 0), span=(1, 5), flag=wx.EXPAND|wx.LEFT|wx.RIGHT, border=5)

        # input entries
        sizer.Add(wx.StaticText(panel, label="Input"), pos=(4, 0), flag=wx.TOP|wx.LEFT, border=5)
        self.tc_send = wx.TextCtrl(panel)
        self.tc_send.Enable(False)
        sizer.Add(self.tc_send, pos=(5, 0), span=(1, 4), flag=wx.EXPAND|wx.LEFT|wx.BOTTOM, border=5)
        self.button_send = wx.Button(panel, label="Send")
        self.button_send.Enable(False)
        sizer.Add(self.button_send, pos=(5, 4), flag=wx.EXPAND|wx.RIGHT, border=5)
        self.button_send.Bind(wx.EVT_BUTTON, self.send)

        # make first column and 3rd row growable
        sizer.AddGrowableCol(1)
        sizer.AddGrowableRow(3)

        panel.SetSizerAndFit(sizer)

    def disconnect(self, event):
        global connectionState
        self.clientSocket.close()
        connectionState = 0

        if event != wx.EVT_BUTTON:
            wx.MessageBox(’Connection lost!’, ’Error’, wx.OK|wx.ICON_ERROR)

        # update UI element statuses
        self.button_connect.SetLabel("Connect")
        self.tc_ip.Enable(True)
        self.tc_port.Enable(True)
        self.tc_send.Enable(False)
        self.tc_send.SetValue("")
        self.button_send.Enable(False)

        # set focus on the ip field
        wx.Window.SetFocus(self.tc_ip)

    def connect(self, event):
        global connectionState

        # disconnect when a socket is connected
        if connectionState:
            self.disconnect(wx.EVT_BUTTON)
            return

        self.clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # set timeout in case a wrong ip is used (in seconds)
        self.clientSocket.settimeout(2)
        ip = self.tc_ip.GetValue()
        port = int(self.tc_port.GetValue())

        try:
            self.clientSocket.connect((ip, port))
        except socket.timeout:
            wx.MessageBox(’Timeout while connecting, verify the IP and port.’, ’Error’, wx.OK|wx.ICON_ERROR)
            return
        except Exception, e:
            wx.MessageBox(’Could not connect to socket, is the app running?’, ’Error’, wx.OK|wx.ICON_ERROR)
            return

        # start the receieveing thread
        connectionState = 1
        self.sct = SocketClientThread(self.clientSocket, self)
        self.sct.start()

        # update UI element statuses
        self.button_connect.SetLabel("Stop")
        self.tc_ip.Enable(False)
        self.tc_port.Enable(False)
        self.tc_send.Enable(True)
        self.button_send.Enable(True)

        # set focus on the send field
        wx.Window.SetFocus(self.tc_send)

    def onKey(self, event):
        global connectionState
        k = event.GetKeyCode()
        if k == wx.WXK_RETURN:
            if not connectionState and wx.Window.FindFocus() == self.tc_ip:
                # try to connect if not yet connected
                self.connect(wx.EVT_BUTTON)
            elif connectionState and wx.Window.FindFocus() == self.tc_send:
                # send data to device
                self.send(wx.EVT_BUTTON)
        elif k == wx.WXK_UP:
            # navigate up in previous commands
            self.navigateCommands(wx.WXK_UP)
        elif k == wx.WXK_DOWN:
            # navigate up in previous commands
            self.navigateCommands(wx.WXK_DOWN)
        else:
            event.Skip()

    def navigateCommands(self, event):
        global sentCommandHistory, sentCommandHistoryId
        if not len(sentCommandHistory):
            return
        if event == wx.WXK_UP and sentCommandHistoryId > 0:
            sentCommandHistoryId = sentCommandHistoryId - 1
            self.tc_send.SetValue(sentCommandHistory[sentCommandHistoryId])
        elif event == wx.WXK_DOWN and sentCommandHistoryId < len(sentCommandHistory) - 1:
            sentCommandHistoryId = sentCommandHistoryId + 1
            self.tc_send.SetValue(sentCommandHistory[sentCommandHistoryId])
        elif event == wx.WXK_DOWN and sentCommandHistoryId < len(sentCommandHistory):
            sentCommandHistoryId = sentCommandHistoryId + 1
            self.tc_send.SetValue("")

    def send(self, event):
        global sentCommandHistory, sentCommandHistoryId
        data = self.tc_send.GetValue()
        if data != "":
            self.sct.send(data)
            sentCommandHistory.append(data)
            sentCommandHistoryId = len(sentCommandHistory)
            self.tc_send.SetValue("")
            self.output.AppendText("> " + data + "\n")

    def updateOutput(self, event):
        global dataBuffer
        self.output.AppendText("< " + dataBuffer + "\n")
        dataBuffer = ""

if __name__ == '__main__':
    app = wx.App()
    SocketClientUI(None, title="Python Terminal Client")
    app.MainLoop()
```

Listing 21: Python source code