ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

**Semester Thesis**

# GasMobile: Operating Gas Sensors with Mobile Phones

**Silvan Sturzenegger**
**ssilvan@ee.ethz.ch**

**Prof. Dr. Lothar Thiele**
**Computer Engineering Group**

**Supervisors: David Hasenfratz**
**Dr. Olga Saukh**

**September 2011 – December 2011**

**Dept. of Information Technology and Electrical Engineering**
**Swiss Federal Institute of Technology (ETH) Zurich**

**Abstract**

The objective of this semester project is to attach a small ozone sensor to a smartphone via USB, so the ozone measurements can be read from the sensor with an Android application. To do this, the linux kernel has to be modified to enable USB host mode, and an application has to be written to control the sensor and take the measurements. Then the influence of wind on the ozone measurements is observed, to get an idea of the impact it has in such a handheld application.

# Contents

# 1 Introduction

## 1.1 Problem Statement

The goal of this work is to find out if it is feasible to attach a USB sensor to a smartphone, and control it with an Android application. It is divided into three parts:

- **Activating USB host mode:** To attach the sensor, which is a USB device, the smartphone has to support USB host mode. The used HTC Hero has a chipset which supports USB host mode, but only the device mode is enabled in the kernel provided by HTC. But there are patches available to enable it, so the first needed step is to compile a custom kernel with the necessary host mode enabled.
- **Android Application:** Once the phone supports USB host mode, an Android application has to be written, which is able to communicate with the sensor. With this application, the user can control the sensor and take measurements.
- **Measurements:** Since a mobile sensor can be exposed to movement and wind, the effects of increased airflow over the sensor is observed by taking measurements.

## 1.2 Choosing the Android Distribution

To do the development, an Android distribution had to be chosen. Apart from the stock Android distributed by the vendor, there are several other Android distributions available. CyanogenMod and VillainROM are two popular examples. For

this project CyanogenMod was chosen, because it has a lot of publicly available documentation[1], and all of its source code is readily available on github[2, 3]. A lot of these so called custom ROMs are already rooted, meaning that the user has superuser privileges. This is certainly needed for this project, since custom kernels need to be installed, kernel modules have to be inserted to activate the host mode etc.

# 2 Activating USB Host Mode

## 2.1 USB Overview

USB (universal serial bus) is a host based bus technology. This means that every bus needs a host controller, which is the master and coordinates all the communication. Multiple slave devices can then be connected to the bus.

Traditionally, cell phones have been slave devices which are connected to computers (the host) to sync data etc. Now that the phones become ever more powerful, they start to have host capabilities themselves to support the USB On-The-Go (OTG) standard. This enables the user to connect peripherals (e.g. mass storage devices, cameras) to the phone without the use of an additional computer. The USB-OTG capable controller is able to switch from the device to the host functionality as required.

Using this functionality, it should be possible to connect the ozone sensor (with an RS232↔USB converter) to the USB-OTG capable smartphone.

## 2.2 HTC Hero Smartphone

For this project, an HTC Hero smartphone is used, which is one of the earlier Android models. This phone uses a Qualcomm MSM7200A chipset[4], which supports USB OTG[5]. The problem is, that the Android kernel supplied by HTC does not have the On-The-Go capability built in, they only provide USB device drivers. Since there is no publicly available documentation for this chipset, it is difficult to write a USB driver from scratch. However, Andrew de Quincey has found out that the MSM7200A chipset uses the same USB controller hardware as some Freescale chips. For these other chips there exists a host driver, which he was

able to adapt so it works for the HTC Hero[6].

These host driver patches he wrote apply to the older 2.6.27 kernel, which is for Android 1.5. In the meantime HTC released a newer kernel, based on the 2.6.29 tree. The recent distributions of the CyanogenMod custom ROM use a modified version of this newer kernel, so it would be nice to apply the patches to this updated kernel.

The basic USB host driver patch only needed some minor changes so it would apply cleanly. However, a big part of the patch added the feature to make the device driver unloadable, so it could be exchanged with the host driver on the fly, as planned in the USB-OTG standard. Unfortunately, these parts of the kernel changed quite a bit, and it would need a lot more work to get this working again. Since the main feature (the host driver) worked, the project was continued without applying the rest of the patch.

The main disadvantage of not having the USB device driver available is that the phone cannot be connected to the computer to sync data, connect to the debugger etc. To do this, the kernel needs to be replaced with the unpatched one and the phone has to be rebooted. Most of the disadvantages can however be circumvented. Data access can be done through SSH over wireless LAN, or the SD-card could be removed to access it. To develop the application, the emulator of the Android SDK can be used, which even provides the capability to connect a serial port of the emulator to a physical one, so the whole communication with the sensor can be done within the emulator.

## 2.3 Lack of Power in Host Mode

A normal USB host controller has to provide enough power on the 5V line to at least power a low power peripheral. This means it would have to provide one unit load, which is 100mA of current at 5V[7]. Since the HTC Hero was not designed for host mode, the controller lacks the ability to power the USB port, even low power ones. (Attempts to provide it have failed[8].) This means that the peripherals have to be powered externally.

To test the host mode, a USB hub was used. There are hubs which use the power provided by the upstrem port from the host controller to power its devices, which is not suited for our purposes. Therefore an externally powered hub was used, which uses its own power supply to power the peripherals. To connect the hub to the phone, a non-standard cable is needed. Both devices have a USB Mini-B receptable, for which there is no cable, since it would allow two hosts to be connected (which could probably cause damage since both power lines would be connected). Therefore, two standard A↔Mini-B cables were connected with a simple gender changer, resulting in the desired non-standard Mini-B↔Mini-B cable.

First attempts were unsuccessful, until it was realized that the hub chip has to be powered from the upstream port, even when the external power supply is plugged in. This enables the hub to be operated like a bus powered hub when its

**Figure 2.1:** *The test setup with the self-powered hub. Connected to the phone are a keyboard, a mouse and the ozone sensor.*

power supply is disconnected, but unfortunately that doesn't work in our case. So a small modification was made, by simply adding a cable from the external 5V power supply to the 5V line of the upstream port, which then powers the hub chip.

Later, when only the ozone sensor without the hub is connected, a battery pack was added to provide 5V to the sensor and RS232↔USB converter.

## 2.4 Other Drivers

Once the USB host driver was available, the other drivers for all the connected devices had to be compiled in as well. To test the functionality, the USB HID (human interface device) driver was added, which enables keyboards and mice. Once the kernel modules for the host mode and HID support were loaded, a keyboard and a mouse could be connected to the powered hub, which is immediately recognised by the operating system and usable. Surprisingly, when moving the mouse a pointer appears and can be used like on a normal computer.

For connecting the sensor, the driver for the RS232↔USB chip is required. After compiling and inserting the kernel module, the serial device shows up on the phone as /dev/ttyUSB0.

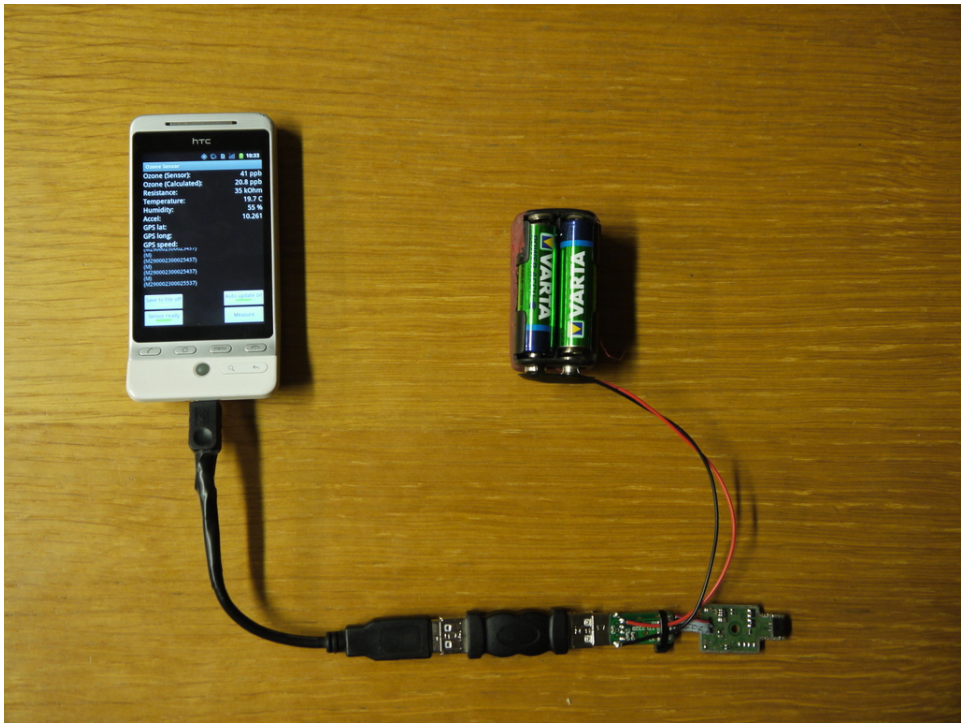



**Figure 2.2:** *The final sensor setup, where the sensor is directly connected to the phone. The RS232↔USB converter and the sensor are powered by the attached battery pack.*

# 3
# Ozone Sensor

The ozone sensor used is a MiCS-OZ-47 from e2v technologies. It has two ozone, one temperature and one humidity sensor built in. The sensor can be controlled via an RS232 connection, with a custom communication protocol.

To connect the sensor to the PC or smartphone via USB, a small RS232 $\leftrightarrow$ USB converter is attached to the sensor board. Specifically, a TTL-232R-PCB from FTDI is used[9], which is a compact PCB directly attached to a USB type-A connector. The driver needed for this converter is already available in the linux kernel sources, so it just has to be enabled when building the custom kernel.

During normal operation, only the first of the two ozone sensors is active. Since the measured values can drift over the lifetime of the sensor, the second one is only powered on from time to time to compare the measured values of the sensors. Because the second sensor is only active for a fraction of the time, its values can be used to correct the drift of the first one.

## 3.1 Communication Protocol

To communicate with the sensor, a simple, custom protocol is used. All the bytes are encoded as ASCII characters. Every command and response is surrounded by curly braces, $\{command\}$ $\{response\}$. The first byte of the string within the curly braces of the response is always the corresponding function code. To take the measurements, only the functions in Table 3.1 are needed, which generate the responses in Table 3.2. The rest of the functions are covered in the datasheet of the sensor[10].

To encode hexadecimal numbers, the following scheme is applied: Every nibble is encoded as one byte in the transmitted string. To every nibble 0x30 (decimal

| | |
|---|---|
| {A} | Get a diagnostics message |
| {S} | Activate auto updating |
| {M} | Get a measurement |

**Table 3.1:** *Commands to send to the sensor.*

| | |
|---|---|
| {Axy} | x and y represent the status of sensor 1 and 2.<br>0: Sensor is OK<br>?: Sensor is defect. |
| {S} | Auto updating is enabled. |
| {Mxxxxxxxxxxxxxx} | Byte 0-1: Ozone value of sensor 1<br>Byte 2-3: Ozone value of sensor 2<br>Byte 4-6: Resistance of sensor 1<br>Byte 7-9: Resistance of sensor 2<br>Byte 10-12: Temperature<br>Byte 13-14: Humidity |

**Table 3.2:** *Responses from the sensor.*

48, ASCII character '0') is added, which means that the numbers are represented as ASCII values in the range from '0' to '?'. For example 0x5A would be encoded to the bytes 0x353A, or as ASCII characters '5:'.

## 3.2 Ozone Concentration Calculations

The ozone concentration can either be read from the measurement response, or calculated from the resistance, temperature and the calibration values.

The resistance values are integers with the unit k$\Omega$. The humidity value is an integer representing the relative humidity as a percentage.

The temperature value has to be corrected to get the real temperature. The value sent in the measurement string is an integer in the range 0-1638 ($-40\,^{\circ}\mathrm{C}$ to $123.8\,^{\circ}\mathrm{C}$). The corrected temperature is then

$$T = \frac{T_{measured}}{10} - 40.$$

The ozone concentration can be calculated from the resistance, the temperature and some calibration coefficients. First, the measured resistance has to be temperature corrected:

$$R_{@25\,^{\circ}\mathrm{C}} = R_{@T}\, e^{kT(T-25\,^{\circ}\mathrm{C})} \tag{3.1}$$

where kT is the temperature coefficient, which is determined during calibration.

The response curve of the sensor to the ozone concentration is quasi-linear and is approximated with a third order polynomial. The ozone concentration can then be determined with the polynomial

$$Ozone[ppb] = X_3 R_{@T}^3 + X_2 R_{@T}^2 + X_1 R_{@T} + X_0. \tag{3.2}$$

The coefficients $kT$ and $X_{0-3}$ are stored in the EEPROM of the sensor and can be read to calculate the concentration on the smartphone itself.

# 4 The Android Application

After adding the USB host driver to the kernel, the phone was ready. So the development of the application to control the ozone sensor could be started.

## 4.1 Emulator Setup

Normally the application under development can be tested and debugged directly on the phone. To do this, the phone has to be connected via USB to the computer, so the debugger has direct access to the phone. Since this application requires that the USB controller is in host mode, it is not possible to debug it directly over USB.

The Android SDK includes an emulator, which has most of the functions a real phone has, including providing fake GPS locations, emulating phone calls etc. The most important feature for this application is the ability to forward the serialports of the emulator to arbitrary ports on the host machine. This can be done by adding the `"-qemu -serial /dev/serialdevice"` switch while launching the emulator, where `/dev/serialdevice` is the path to the serial port on the host computer.

For the first attempt to connect the sensor to the emulator, the emulator was directly connected to the `"/dev/ttyUSB0"` device, which is the serialport of the sensor. Unfortunately this didn't work. It was suspected that the serialport configuration used in the Android application didn't apply to the host computer, which prevented the communication. To test whether the serial port forwarding really worked, the traffic instead was forwarded to minicom, where the connection worked in both ways.

To now establish the connection with the sensor, a python script was created to correctly configure the sensors serial port, and again forward all the data be-


**Figure 4.1:** *Diagram of the emulator setup.*

tween the emulator and the sensor, as illustrated in Figure 4.1. This solution finally worked and even allowed to intercept and print all the serial data, which could be used to detect bugs and mistakes in the low level communication.

## 4.2 Debugging Over SSH

There is another possibility to debug the application, this time not on the emulator, but on the real phone. When using the emulator, the debugger connects to it over TCP/IP on port 5554/5555 (default values for the first connected emulator). When an SSH tunnel is created from the host machine to the phone, forwarding the ports 5554 and 5555, the debugger is able to directly connect to it. This method was only successfully tested but not used, because it is very slow and using the emulator is so convenient.

## 4.3 Application Design

The following features should be implemented and integrated in a simple user interface:

- Establish a serial connection with the sensor.
- Read measurements from the sensor (periodically).
- Get the current GPS location, so the position of the measurement is known.
- Access the accelerometers, for possible movement analysis.
- Log all the data to a file on the SD-card for further offline processing.

### 4.3.1 Serialport API

Android itself does not provide an API to access serial ports at this time, but there is a project called android-serialport-api which implements this ability[11]. It provides functions for listing all available serial ports and configuring them (setting the baudrate, stop bits etc). For reading and writing to the serial port, it provides

standard Java InputStream and OutputStream interfaces. There are sample applications included which demonstrate the capabilities, and provide a good starting point for developing the sensor application.

### 4.3.2 GPS and Accelerometer Data

For accessing the GPS and accelerometer data, there are APIs provided by Android. There just have to be two listeners implemented, which listen to sensor changes and update the application accordingly.

### 4.3.3 User Interface Design

#### Main Screen

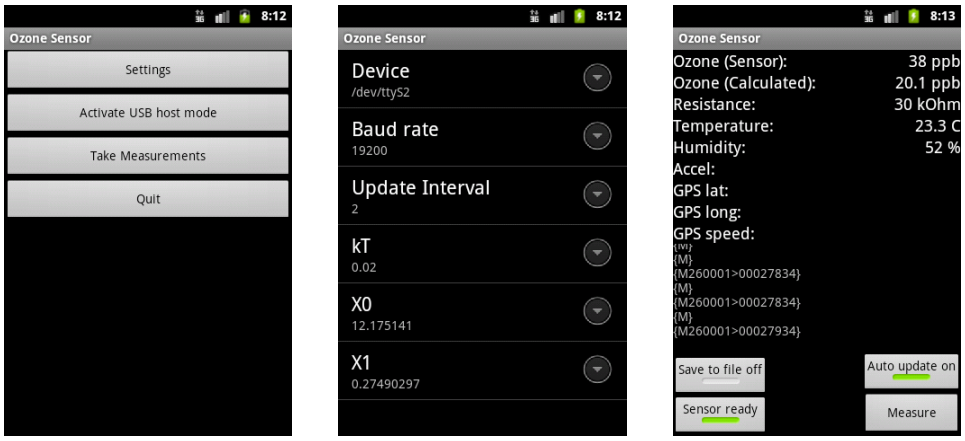
The main menu (Figure 4.2a) is the first screen to appear when the application is started. It features four buttons to access the main functionalities.

- **Settings:** Access the settings, to change permanent configuration values.
- **Activate USB host mode:** Activate the USB host mode by loading the necessary kernel modules and load the RS232 $\leftrightarrow$ USB converter driver.
- **Take Measurements:** Go to the measurement screen.
- **Quit:** Quit the application

#### Settings

The settings screen (Figure 4.2b) lets the user adjust different configuration values of the application, which are stored permanently on the phone. It is accessed by pressing the *Settings* button in the main menu.

- **Device:** The sensors serial device. It only appears when host mode is active and the device driver is loaded. The sensor should appear as device */dev/ttyUSB0*.
- **Baud Rate:** The baudrate of the RS232 connection. The ozone sensor requires a rate of 19200.
- **Update Interval:** The interval in which the measurements are taken when auto updating is turned on.
- **kT, X0, X1:** The calibration values to calculate the ozone concentration from the measured resistance and temperature. These can be set indepently from the calibration values on the sensor itself.




**(a)** *Main menu* **(b)** *Settings screen* **(c)** *Measurement screen*

**Figure 4.2:** *The different user interface screens of the application.*

### Measurement Screen

In the measurement screen (Figure 4.2c), accessed with the *Take Measurements* button in the main menu, the sensor can be controlled and measurements are read. There are 4 buttons to control the measurements.

- **Save to file:** Enables the logging function. When pressed, all the measured data is logged to a file on the SD-card. The file is written to `/sdcard/measurements/` and has the current date and time in the filename.
- **Init sensor:** When pressed, the sensor is initialized. A diagnostics message is requested, and the sensor is set to auto mode, so it regularly updates its measurements. If it is successful, the button turns green and measurements can be taken.
- **Measure:** Take a single measurement.
- **Auto update:** Let the application periodically update the measurements, with the interval specified in the preferences.

The first five lines of the displayed data is the measured sensor data. The value labeled with *Ozone (Calculated)* is the ozone concentration calculated with the calibration settings in the application preferences, whereas *Ozone (Sensor)* is the one reported from the sensor with its own calibration coefficients.

The other four lines are data reported by the phone. The acceleration value is the absolute phone acceleration, calculated from the x,y,z components. If available, the current GPS location is displayed, as well as the movement speed reported by the GPS.

# 5 Measurements

## 5.1 Power and Resource Consumption

### 5.1.1 Power Consumption of the Sensor Hardware

Since the sensor is battery operated and should be mobile, there is a tradeoff between operating time and battery size. To operate the sensor, a battery pack with four AAA NiMH batteries was chosen. Depending on the charge in the batteries, the voltage is more or less at 5V, which is the required target voltage provided by the USB port. To estimate the power consumption of the sensor, the combined current of the RS232↔USB converter and ozone sensor is measured and listed in Table 5.1.

When the sensor is power on, the sensing elements are initially overheated to converge faster to a relevant measurement. This leads to a 3mA current increase during the first minute.

There was no difference detected when comparing the power consumption while idling and taking periodic measurements, where the data is exchanged over RS232 and USB. There is however a 9mA increase in the current when the USB cable is connected.

The used batteries have a nominal capacity of 2500mAh. To get an idea of how long they would last, the capacity is divided by the drawn current:

$$\frac{2500\text{mAh}}{43\text{mA}} = 58.14\text{h} \tag{5.1}$$

This obviously isn't an accurate equation, because the actual capacity could be different or the sensor doesn't work anymore when the voltage drops below a certain level. But it gives the order of magnitude of how long it is possible to power

the hardware.

| | |
|---|---|
| USB disconnected | 34mA |
| USB connected | 43mA |
| Ozone sensor overheating (After powering on) | +3mA |

**Table 5.1:** *Power consumption of the ozone sensor and RS232↔USB converter.*

### 5.1.2 Resources Used by the Android Application

If the Android application uses a lot of resources on the phone, it can dramatically reduce its operating time. Therefore the used system memory and CPU is observed and listed in Table 5.2. The ∼5.5MB unique to the application are more meaningful, because the ∼25MB shared with other processes would be used anyway.

There is a rather large 10% difference in CPU consumption when the display is off. The application is still running and the measurements are taken, but the whole screen doesn't have to be updated anymore. Since the application displays the absolute acceleration which is updated in a short interval, this has a big impact. Since the acceleration is of no significant interest to the enduser, it should be easy to reduce the CPU consumption by disabling this feature.

| | |
|---|---|
| System memory (Unique to the application) | ∼5.5MB |
| System memory (Shared with other processes) | ∼25MB |
| CPU (Display on) | ∼15% |
| CPU (Display off) | ∼5% |

**Table 5.2:** *Phone resources used while running the application.*

## 5.2 Wind Measurements

The sensor is now attached to a smartphone and the user who takes the measurements can move around. When the sensor is moving through the air or being exposed to environmental effects like wind, the airflow around the sensor changes. To observe the effects of this changed airflow, measurements are taken under different conditions and the sensor values are checked for changes.

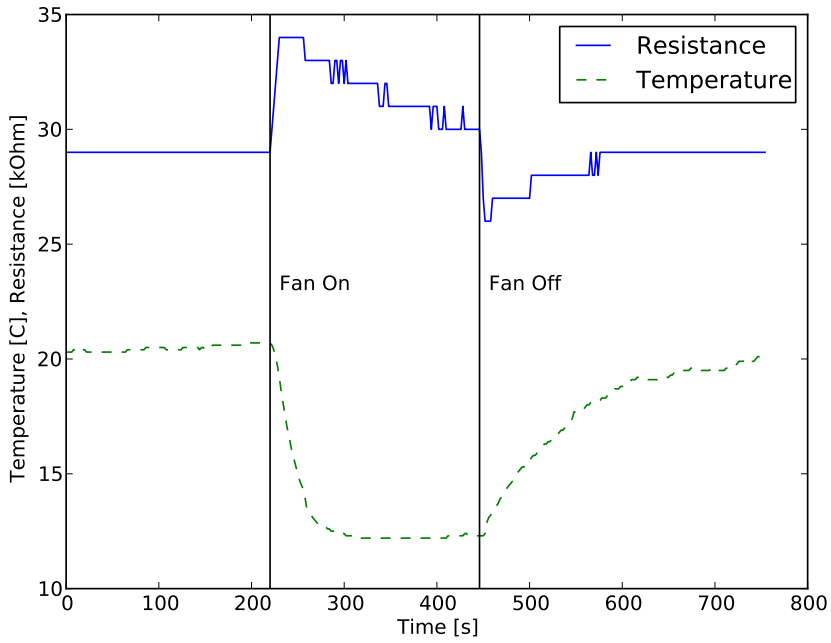



**Figure 5.1:** *The influence of wind on the ozone sensor. As soon as the fan is switched on or off, there is a jump in the measured resistance.*

### 5.2.1 Measurement Method

To measure the effects on the sensor, it is set up inside a room where the ozone concentration should be constant. To simulate wind which changes the airflow around the sensor, a desk fan is used. When the fan is switched on, the relative changes of the sensor values can be observed.

### 5.2.2 Ozone Sensor

The sensor is set up right in front of the desk fan, to maximize the airflow. First, a series of measurements is taken for four minutes when the desk fan is off and the surrounding air is still, to have a reference point. Then the fan is turned on for four minutes, where changes in the sensed values can be observed. Finally the fan is turned off again, to see the impact of reversing the effect. The whole measurement series is logged to a file using the Android application. Figure 5.1 shows the measured resistance and temperature values.

The ozone concentration can be calculated using these two measured values, using the calibration values and the formulas 3.1 and 3.2.

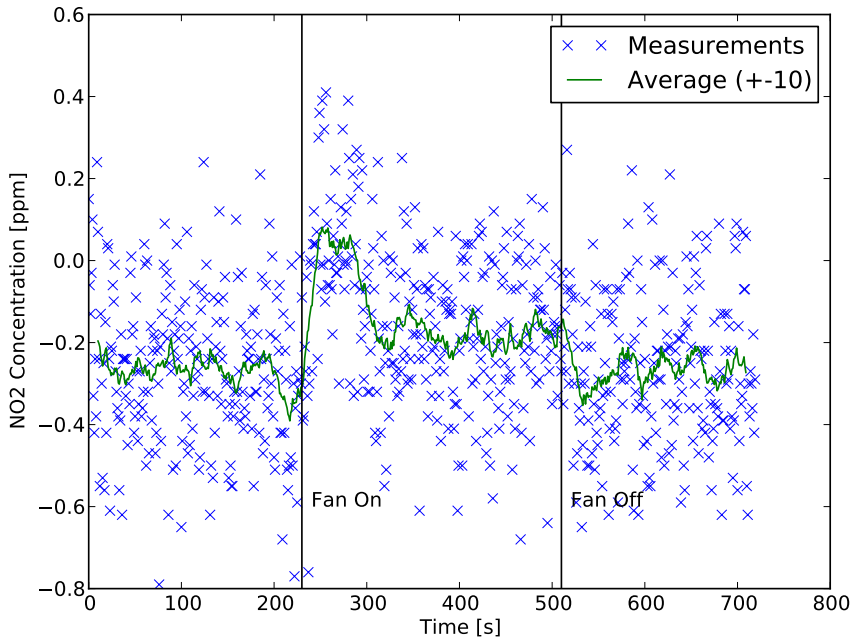



**Figure 5.2:** *The influence of wind on the $NO_2$ measurements. An offset can be observed when the fan is turned on and the measurements return to their previous values when turning it off again.*

### 5.2.3 $SO_2$ and $NO_2$ Sensors

Two additional gas sensors were tested, to have additional measurements and a broader overview of the wind influence. The tested sensors are for measuring $NO_2$ and $SO_2$ concentrations. These were however not connected to the smartphone, the measurements were taken on a computer.

The exact same setup was used as with the ozone sensor, and the resulting measurements are displayed in Figures 5.2 and 5.3.

## 5.3 Discussion

From the results of these tests can be seen clearly that the wind has an effect on the measurements.

Especially when looking at the ozone measurements, right at the moment when the fan is switched on there is a sudden jump in the measured resistance. After the jump the resistance starts to drop again, but it is always above the level measured when in still air. Likewise when the fan is switched off, there is a jump in the negative direction and after about two minutes it is at the same level as before.

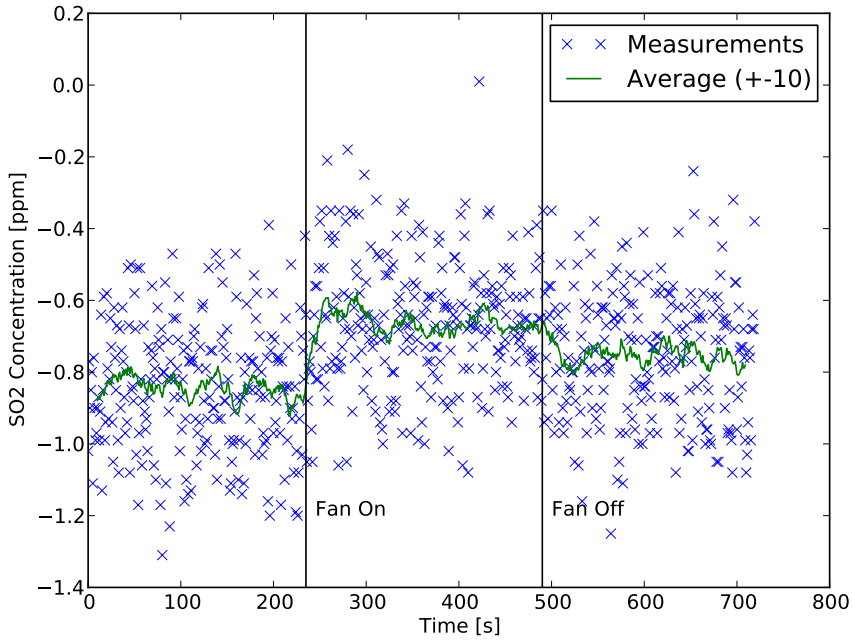



**Figure 5.3:** *The influence of wind on the $SO_2$ measurements. As with the $NO_2$ sensor, there are jumps when the fan is turned on and off.*

One interesting point is the fact that the measured temperature decreases by about $8°\mathrm{C}$. Since the introduced wind shouldn't change the actual temperature of the sensor board, it is surprising that the measured value changes this much. There is nothing mentioned of such a dramatic effect in the temperature sensors datasheet. One explanation could be that the desk fan circulates the air in the room and cooler air from the floor reaches the sensor, or that the wind increases the dissipation of local heat sources on the sensor board itself. But it is unlikely that these effects account for such a large heat differential.

Since the used ozone sensor was not calibrated, only raw values of the sensor resistance and temperature were observed. An interesting additional measurement would be to use a calibrated sensor, and check if the temperature and resistance offsets would cancel each other out with the correct calibration coefficients. But there certainly has to be some disturbance in the measurement, because right at the moment when the wind is introduced there is a sudden jump in the resistance, whereas the temperature only starts to drop gradually. So even if the effect would be cancelled out in a constant wind environment, wind bursts would still falsify the measured concentration.

When looking at the $\mathrm{NO}_2$ and $\mathrm{SO}_2$ measurements the effect is less visible, because the variance of the individual measurements is high. When the values are averaged around a window of $\pm 10$, the jumps are more visible, and there is an offset introduced in both sensors when the airflow is high.

# 6
# Conclusion and Future Work

From this work can be concluded that connecting sensors to a smartphone is certainly doable. Since the used sensors have rather simple communication protocols and use serial connections, it should also be easy to extend the application to support other sensors as well.

Another important aspect of this work is that only low cost hardware was used. This is especially important when participatory sensing wants to be done, and sensors should be distributed in larger numbers.

The introduced mobility when the sensor can be controlled with a smartphone only and there is no need to carry around equipment like computers and power supplies offers numerous advantages. It is for example easier to calibrate the sensor near a fixed measurement station, or further effects of mobility and wind can be investigated.

There is a lot that can still be done:

- At the moment the host driver on the HTC Hero cannot be exchanged with the device driver, which would be unacceptable for a regular user. Some newer smartphones however have proper USB-OTG support, so the application could be extended to make use of this feature.
- Add support for additional sensors.
- To better understand the influence of wind on the measurements, further measurements have to be done.
- The sensor hardware can be improved. Everything could be integrated in a small box and the power supply could be replaced by a regulated one, so the voltage drop of the batteries is eliminated.

# A Developer How-To

## A.1 Android SDK and Eclipse Plugin

To build the Android application, connect to the phone etc, the Android SDK is needed. It is available at `http://developer.android.com/sdk/index.html`. If development is done with Eclipse, the ADT Eclipse plugin can be used, which integrates a lot of the SDK directly into Eclipse and is available at `http://developer.android.com/sdk/eclipse-adt.html`.

## A.2 Patching and Building the Kernel

These instructions apply to version 7.1.0 of CyanogenMod.

First the kernel sources for the Android kernel are needed[3]. For the used CyanogenMod 7.1, the tag flykernel-12a has to be checked out.

```
git clone https://github.com/erasmux/\
hero-2.6.29-flykernel.git
cd hero-2.6.29-flykernel/
git checkout flykernel-12a -b usb-host
```

Then apply the USB host patch.

```
patch -p1 < usb_host_patch
```

To compile the kernel, an ARM EABI toolchain is needed, for example the Code Sourcery toolchain from `https://sourcery.mentor.com/sgpp/`

`lite/arm/portal/subscription3053`. Additionally, a kernel configuration file is required. Either the already modified version is used, or the configuration can be extracted from the phone.

```
adb pull /proc/config.gz ./
gunzip config.gz
```

Add the new toolchain to the path variable and set the needed flags.

```
export PATH=$PATH:~/path/to/toolchain/bin/
export ARCH=arm
export CROSS_COMPILE=arm-none-eabi-
```

Configure the kernel (or load the supplied configuration), and build it. Also install the built modules to a local folder.

```
make menuconfig
make -j3
export INSTALL_MOD_PATH=./modules
make modules_install
```

## A.3 Installing the Modified Kernel

To create a boot image, an old one from the phone is required. It can be extracted with the following commands

```
adb shell
cat /dev/mtd/mtd2 > /sdcard/original_boot.img
exit
adb pull /sdcard/original_boot.img ./
```

Now the new boot image has to be created.

```
mkdir temp
cp original_boot.img temp/boot.img
cd temp/
extract-kernel.pl boot.img
extract-ramdisk.pl boot.img
rm boot.img-kernel
cp ../arch/arm/boot/zImage boot.img-kernel
mkbootfs boot.img-ramdisk | gzip > ramdisk-boot
mkbootimg --kernel boot.img-kernel --ramdisk \
ramdisk-boot --cmdline "no_console_suspend=1 \
console=null" -o boot.img --base 0x19200000
mv boot.img ../
cd ../
rm -r temp
```

Now everything is ready to be copied to the phone. Replace the 2.6.29.6-custom-flykernel-12a with the appropriate name if it is different.

```
adb push boot.img /sdcard/
adb shell mount -o remount,rw /system
adb push modules/lib/modules/\
2.6.29.6-custom-flykernel-12a/kernel \
/system/lib/modules/2.6.29.6-custom-flykernel-12a
adb push modules/lib/modules/\
2.6.29.6-custom-flykernel-12a/kernel/drivers/net\
/wireless/tiwlan1251/wlan.ko /system/lib/modules/
```

Finally, the boot image has to be written over the old one.

```
adb shell
cat /dev/zero > /dev/mtd/mtd2
flash_image boot /sdcard/boot.img
reboot
```

If everything worked correctly, the modified kernel should now be installed and working.

## A.4 Secure Shell Access

It is useful to have SSH to access the phone when the kernel is modified and USB device mode is unavailable (connecting with `adb` won't work). An SSH server is already installed on the phone, it just has to be configured.

On the computer, if not already done, create an RSA keypair and copy the public key to the phone.

```
ssh-keygen -t rsa
adb push ~/.ssh/id_rsa.pub /sdcard/authorized_keys
```

Then connect to the phone

```
adb shell
mount -o remount,rw /system
```

and configure the SSH server.

```
mkdir /data/dropbear
chmod 755 /data/dropbear
mkdir /data/dropbear/.ssh
chmod 700 /data/dropbear/.ssh
cp /sdcard/authorized_keys /data/dropbear/.ssh/
```

```
chown root: /data/dropbear/.ssh/authorized_keys
chmod 600 /data/dropbear/.ssh/authorized_keys
dropbearkey -t rsa -f /data/dropbear/\
dropbear_rsa_host_key
dropbearkey -t dss -f /data/dropbear/\
dropbear_dss_host_key
dropbear -s
```

To run the server by default, modify the `/etc/init.local.rc`.

```
echo "# start SSH server on boot" \
>> /etc/init.local.rc
echo "service sshd /system/xbin/dropbear -s" \
>> /etc/init.local.rc
echo "    user root" >> /etc/init.local.rc
echo "    group root" >> /etc/init.local.rc
echo "    oneshot" >> /etc/init.local.rc
```

Now everything should be ready to connect to the phone.

```
ssh root@IP
```

## A.5 Building the Application

The easiest way to build the application is to use Eclipse with the Android Developer Plugin. A complete Eclipse project is available, where everything is configured already. To install the application when USB is not available, it can be copied to the SD-card and installed from there with the file manager on the phone.

# References


[1] "CyanogenMod documentation." http://wiki.cyanogenmod.com/index.php?title=Main_Page. [Online; accessed 21-December-2011].

[2] "CyanogenMod source code." https://github.com/CyanogenMod/android. [Online; accessed 21-December-2011].

[3] "HTC Hero kernel source code for CyanogenMod." https://github.com/erasmux/hero-2.6.29-flykernel. [Online; accessed 21-December-2011].

[4] "HTC Hero Specifications." http://wiki.cyanogenmod.com/wiki/HTC_Hero_(28GSM)29. [Online; accessed 14-December-2011].

[5] "MSM7200A Chipset." http://www.datasheetpro.com/node/13806. [Online; accessed 14-December-2011].

[6] "HTC Hero USB host mode." http://adq.livejournal.com/95689.html. [Online; accessed 14-December-2011].

[7] "USB 2.0 Specifications." http://www.usb.org/developers/docs/usb_20_101111.zip. [Online; accessed 14-December-2011].

[8] "Bus power attempt on HTC Hero." http://adq.livejournal.com/100591.html. [Online; accessed 14-December-2011].

[9] "Datasheet for RS232 $\leftrightarrow$ USB converter." www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_TTL-232R_PCB.pdf. [Online; accessed 22-December-2011].

[10] "Datasheet for the ozone sensor." http://www.e2v.com/e2v/assets/File/sensors_datasheets/Metal_Oxide/mics-oz-47.pdf. [Online; accessed 21-December-2011].

[11] "android-serialport-api." https://code.google.com/p/android-serialport-api/. [Online; accessed 14-December-2011].