DFRobot project



Contents

Introduction 3

Raspberry Pi B+ configuration 4

Backing up the SD card image of the Raspberry Pi 4

Mounting the Raspberry Pi SD card in Ubuntu 4

Connecting the Raspberry Pi to the MacBook via the ethernet cable 4

Network configuration in /etc/network/interfaces 5

Wifi configuration in /etc/wpa\_supplicant/wpa\_supplicant.conf 5

Connecting the Raspberry Pi to the wireless network (no Ethernet cable) 6

Logging into the Raspberry Pi 6

Raspberry Pi file sharing 6

Raspberry Pi camera module 7

Apache webserver port configuration 7

Apache webserver password protection 7

Enable .py scripts with CGI in Apache2 8

Bash scripts versus Python scripts with CGI in Apache2 8

Enabling I2C on the Raspberry Pi 8

Enabling the watchdog on the Raspberry Pi (solution 1, does not work) 9

Installing a custom watchdog on the Raspberry Pi (solution 2, does work) 9

Arduino Uno configuration 11

Measurements 12

Operate the DFRobot using only 3G or 4G 14

# Introduction

This project was started beginning 2015 to get familiar and (even more) enthousiastic about embedded systems, embedded Linux, Robotics. Its fits very well the curriculum of the Fontys FHICT Technology study. Topics addressed are:

* Embedded systems
* Embedded Linux
* Networking
* Web server, webpage development
* Video streaming over internet
* ROS
* Programming C, C++, Python

To get started the following hardware is used:

* DFRobot 4WD Arduino Mobile Platform
* Raspberry Pi Model B+ Computer Board
* Raspberry Pi Camera Module
* DFRduino Uno USB Microcontroller V2.0
* DFRobot Arduino Compatible Motor Shield (2A)
* Lynxmotion Pan and Tilt Kit / Aluminium
* Netgear Wireless N Adapter 150 Mbps (USB) (WNA1100-100PES)

The project will never be ‘finished’ and is perfect for keeping up to date with the latest technologies. At the moment the following phases are foreseen:

**Phase one: feasibility**

In this phase I want to investigate the areas which I see as high risk and which are crucial for success of the project. Better to encounter problems in these areas right away then later in the project.

* Spike to see if it was possible to have low latency (< 2 sec) streaming video from the Raspberry Pi B+ to a standard web browser (no plugins). This is not trivial and a lot of discussions can be found in forums.
* Spike to see how to control the Pi Camera and the Pi I/O pins from a standard web browser. Different techniques can be used and it would be good to use the right one for this project.

**Phase two: Basic system up and running**

* Robot platform controlled from a standard web browser containing streaming video. Status information will also be available.
* Next to that a charging station will be built and it should be possible to manually drive to the driving station and attach.

**Phase three: Adding more features / intelligence**

In this phase I want to add intelligence and features.

* Adding sensors like for measuring dinstance, temperature etc…
* Adding servo’s for grabbing.
* Let the robot create a map of the environment so that it can orientate itself and one can give commands like: go to room X position Y. This is where ROS comes in.
* Let the robot automatically find and attach to its charging station when needed.

# Raspberry Pi B+ configuration

## Backing up the SD card image of the Raspberry Pi

To create an image of the Pi SD card the dd command can be used. Put the SD card in the Mac and identify the 8 GB SD card disk with ‘diskutil list’. The Identifier will be something like ‘disk1’. Then in the dd command one can use /dev/disk1 for the device or for higher speed the (unbuffered) raw disk /dev/rdisk1 which is ok when using the dd command.

sudo dd if=/dev/rdisk1 of=/Users/fhict/ReneB/GitHub/rbakx/DFRobot\_SDcardBackup\_TooLargeForGithub/SDcardBackup20150124.img bs=1m

or to save space:

sudo dd if=/dev/rdisk1 bs=1m | gzip > /Users/fhict/ReneB/GitHub/rbakx/DFRobot\_SDcardBackup\_TooLargeForGithub/SDcardBackup20150124.gz

This will create an image you can use to create a new SD card. This image can be restored on an SD card using again the dd command:  
sudo dd bs=1m if=/Users/fhict/ReneB/GitHub/rbakx/ DFRobot\_SDcardBackup\_TooLargeForGithub/SDcardBackup20150124.img of=/dev/rdisk1

## Mounting the Raspberry Pi SD card in Ubuntu

When the Raspberry Pi SD card is corrupt and connection to the DFRobot is not possible anymore, it might be possible to fix it by changing a file on the SD card. Then it is useful to be able to write Raspbian files when inserting the SD card in the Mac. Because the Raspbian Linux partition cannot be read normally under OS X (except with Fuse for OS X), we make use of Ubuntu and WMware Fusion (VirtualBox did not seem to work here).  
On the Mac, start Ubuntu in VMware Fusion. Insert the SD card in the Mac and eject it under OS X. In VMware Fusion: Ubuntu 64 bit -> Virtual Machine -> USB & Bluetooth -> Connect Apple Internal Memory Card Reader. In Ubunto the memory card should appear. In the file browser it is read only so we need to mount it to be able to acces it in terminal. Open terminal and list the devices with ‘sudo fdisk –l’. There the name of the Linux partition of the SD card can be seen, e.g. ‘/dev/sdb2’. This device can be mounted with ‘sudo mkdir /sdcard’ and then ‘sudo mount /dev/sdb2 /sdcard’. Then the Linux partition can be accessed at ‘/sdcard’ from the terminal. ‘sudo’ can be used to write files if needed. Note that some files cannot be seen as standard user as they are readable only by the root.

## Connecting the Raspberry Pi to the MacBook via the ethernet cable

We want to connect the Pi to the MacBook through the Ethernet cable so we can control the Pi with the keyboard / screen of the MacBook and also provide internet access to the Pi. The latter is done by enabling internet sharing on the MacBook.

* First configure the MacBook Ethernet network settings to static IP 192.168.2.1. It must be this address because this is the fixed IP address which the OS X Mavericks uses when enabling internet sharing. The address 192.168.2.1 address means that the Pi must have a static address in the same subnet, e.g. 192.168.2.2. This is accomplished by setting this in **/etc/network/interfaces**. Also the gateway of the Pi has to be set there and this must be the IP of the MacBook: IP 192.168.2.1.
* Internet sharing from MacBook:  
  When enabling internet sharing on the MacBook the MacBook apparently always uses 192.168.2.1. This is on a different subnet then the Wifi, so the MacBook acts as a router.
* A reboot might be needed before internet sharing is really working. Still the setup seems not very reliable.

## Network configuration in /etc/network/interfaces

For the wlan0 interface (Wifi) DHCP is used. Using DHCP is convenient so the Pi also can operate in other networks, like in a phone’s tethering network! A fixed DHCP address can be supplied by the router, which is convenient so the Pi always has the same IP address which is also needed for port forwarding. This means that the router must assign a static DHCP IP address (192.168.1.42) to the Pi (using its MAC address).  
Below note that for the ethernet interface no DHCP is used but static IP address settings which is convenient because the Ethernet DHCP service on the Mac apparently cannot supply a static DHCP IP address, so finding out the Pi’s IP address is difficult then.

auto lo

iface lo inet loopback

iface eth0 inet static

address 192.168.2.2

netmask 255.255.255.0

gateway 192.168.2.1

allow-hotplug wlan0

iface wlan0 inet manual

wpa-roam /etc/wpa\_supplicant/wpa\_supplicant.conf

iface default inet dhcp

## Wifi configuration in /etc/wpa\_supplicant/wpa\_supplicant.conf

Below note that two access points are specified to allow roaming. Also note the network section for the tethering network of a phone!

ctrl\_interface=DIR=/var/run/wpa\_supplicant GROUP=netdev

update\_config=1

network={

ssid="wifiwifiwifi2"

psk="fill in password here"

# Protocol type can be: RSN (for WP2) and WPA (for WPA1)

proto=RSN

# Key management type can be: WPA-PSK or WPA-EAP (Pre-Shared or Enterprise)

key\_mgmt=WPA-PSK

# Pairwise can be CCMP or TKIP (for WPA2 or WPA1)

pairwise=CCMP

#Authorization option should be OPEN for both WPA1/WPA2 (in less commonly used $

auth\_alg=OPEN

}

network={

ssid="wifiwifiwifi"

psk="fill in password here"

# Protocol type can be: RSN (for WP2) and WPA (for WPA1)

proto=RSN

# Key management type can be: WPA-PSK or WPA-EAP (Pre-Shared or Enterprise)

key\_mgmt=WPA-PSK

# Pairwise can be CCMP or TKIP (for WPA2 or WPA1)

pairwise=CCMP

#Authorization option should be OPEN for both WPA1/WPA2 (in less commonly used $

auth\_alg=OPEN

}

network={

ssid="AndroidAP"

psk="AndroidAP"

# Protocol type can be: RSN (for WP2) and WPA (for WPA1)

proto=RSN

# Key management type can be: WPA-PSK or WPA-EAP (Pre-Shared or Enterprise)

key\_mgmt=WPA-PSK

# Pairwise can be CCMP or TKIP (for WPA2 or WPA1)

pairwise=CCMP

#Authorization option should be OPEN for both WPA1/WPA2 (in less commonly used $

auth\_alg=OPEN

}

Testing internet speed from command line:  
wget -O /dev/null http://speedtest.wdc01.softlayer.com/downloads/test10.zip

## Connecting the Raspberry Pi to the wireless network (no Ethernet cable)

* Configure the file /etc/network/interfaces and the /etc/wpa\_supplicant/wpa\_supplicant.conf like above.
* The Pi will get an IP address assigned by DHCP: 192.168.1.42.

## Logging into the Raspberry Pi

* Connect to Pi via SSH: ssh [pi@192.168.1.42](mailto:pi@192.168.1.42), password ‘raspberry’.
* Install VNC server on Pi and VNC client on MacBook.
* Start up VNC server on Pi with: ‘vncserver: 1’.
* Start up VNC client on MacBook and connect to 192.168.1.42, password ‘raspberr’.

## Raspberry Pi file sharing

At http://4dc5.com/2012/06/12/setting-up-vnc-on-raspberry-pi-for-mac-access/ it is described how to set up file sharing. Netatalk is installed which is an open source AFP (Apple Filing Protocol) file server. Then Avahi is installed which is a zero configuration service discovery protocol. The file ‘/etc/avahi/services/afpd.service’ is created with the avahi settings (a.o. TCP/IP port 548 is specified). Avahi is then started as a deamon which will start automatically after each reboot.

In /etc/netatalk/AppleVolumes.default at the end there is:

~/ "Home Directory"

which is the default folder to share. There you can add an extra path to share like:

/etc "/etc"

/lib “/lib”

Then on the Mac, in finder, press cmd-K (connect to server) and enter the Pi IP address and password. Then the Pi shared folder shows up in the shared section in the Finder sidebar.  
When writing to a Pi folder like ‘/etc’ from the Mac the permission is denied. This is because the user ‘pi’ is not the owner or member of the group of that folder (root is) and the permission is on 755. When writing to ‘/etc’ is needed, on the Pi do: ‘sudo chmod 777 /etc’.

After writing put back permission to 755 with ‘sudo chmod 755 /etc’.

## Raspberry Pi camera module

Follow the instructions at ‘https://miguelmota.com/blog/raspberry-pi-camera-board-video-streaming/’ and follow the instructions at ‘Update: 19 Jan 2014 - Easier way to stream’ or ‘http://petrkout.com/electronics/low-latency-0-4-s-video-streaming-from-raspberry-pi-mjpeg-streamer-opencv/’ to install the MJPG Streamer.

Start the camera and MJPG stream:

$ LD\_LIBRARY\_PATH=/opt/mjpg-streamer/ /opt/mjpg-streamer/mjpg\_streamer -i "input\_raspicam.so –vf –hf -fps 15 -q 50 –ex sports -x 800 -y 600" -o "output\_http.so -p 44445 -w /opt/mjpg-streamer/www" > /dev/null 2>&1 &

Then the stream can be viewed at 192.168.1.42:44445.

The extra ‘> /dev/null 2>&1’ (redirect stderr to stdout) at the end is to make sure the MJPG\_streamer command returns with a prompt. Otherwise it seems to hang (despite the ‘run as background task symbol ‘&’) and when called from CGI Python the Python script hangs.

## Apache webserver port configuration

To set the correct Apache webserver port edit ‘/etc/apache2/ports.conf’ and change port 80 to 44444:

NameVirtualHost \*:44444

Listen 44444

In addition in ‘/etc/apache2/sites-enabled/000-default’ change the first line from <VirtualHost \*:80> to <VirtualHost \*:44444>.

## Apache webserver password protection

See <http://www.debiantutorials.com/password-protecting-a-directory-with-apache-and-htaccess/>.

The DFRobot website is protected by letting the Apache server protect the ‘/var/www’ directory, where the index.html file is located. The file ‘/etc/apache2/sites-enabled/000-default’ is modified to contain:

<Directory /var/www/>

AllowOverride All

</Directory>

This enables ‘.htaccess’ file support in ‘/var/www’.  
A ‘.htaccess’ file is created and placed in /var/www (where the index.html file is located) to protect the complete DFRobot website:

AuthType Basic

AuthUserFile /etc/apache2/.htpasswd

AuthName "Enter password"

Require valid-user

It points to ‘/etc/apache2/.htpasswd’ which contains the password for user ‘reneb’. The command ‘htpasswd -c /etc/apache2/.htpasswd reneb’ can be used to generate a new password for user ‘reneb’.

## Enable .py scripts with CGI in Apache2

Open ‘/etc/apache2/sites-enabled/000-default’ (this is a link to the file ‘default’) and add  
’AddHandler cgi-script .py’ to the section

<Directory "/usr/lib/cgi-bin">  
...  
</Directory>

Then reload Apache’s configuration using ‘sudo service apache2 reload’.

When running Python scripts from Apache2 through CGI the user is ‘www-data’. So scripts and external program run must have the proper group permissions.  
When using the camera module the VideoCore device ‘/dev/vchiq’ is used. The default permissions are:  
crw-rw---T 1 root video 250, 0 Jan 1 1970 /dev/vchiq  
These permissions mean that www-data cannot read from ‘/dev/vchiq’ which will give the ‘failed to open vchiq instance’ error when trying to use the camera from CGI Python. It can be seen that the user group ‘video’ is assigned to ‘/dev/vchiq’.

To be able to run the camera from CGI Python: ‘sudo chmod 666 /dev/vchiq’ works, but the original permissions are reset on reboot.  
So a permanent solution is to make the www-data (www-data is the user under which the Apache2 web server runs) member of the video group. This can be done with: ‘sudo adduser www-data video’.

## Bash scripts versus Python scripts with CGI in Apache2

When using Python CGI the problem is encountered that for each http POST method call the Python interpreter on the Pi must be loaded and the Python script executed. This can take 1 to 3 seconds, depending if video is streaming or not. As this is not an acceptable delay I decided to switch from Python CGI to bash CGI which is much faster.

## Enabling I2C on the Raspberry Pi

Information taken from <http://www.legomindstormsrobots.com/arduino/connecting-arduino-raspberry-pi-i2c/>

Enable the I2C hardware by adding at the end of /etc/modules:  
i2c-bcm2708

i2c-dev

Next install the i2c-tools utility:

‘sudo apt-get install python-smbus’.

‘sudo apt-get install i2c-tools’.

If /etc/modprobe.d/raspi-blacklist.conf exists, comment out the lines  
blacklist spi-bcm2708

blacklist i2c-bcm2708

From kernel version 3.18 on (‘uname –a’ shows kernel version): to use GPIO2 (pin 3) and GPIO3 (pin 5) as SDA and SCL respectively, add ‘dtparam=i2c1=on’ to the end of /boot/config.txt.  
Add the pi user to the i2c group, to do this type 'sudo adduser pi i2c'.

The I2C devicescan be scanned using ‘i2cdetect -y 1’.

To allow usage of the I2C device by www-data:  
‘sudo adduser www-data i2c'.

Finally, in ‘/lib/udev/rules.d/60-i2c-tools.rules’, change  
KERNEL=="i2c-[0-9]\*", GROUP="i2c", MODE="0660"

to  
KERNEL=="i2c-[0-9]\*", GROUP="i2c", MODE="0666"

to have the I2C device readable by the i2c group.  
Then reboot using ‘sudo reboot’.

## Enabling the watchdog on the Raspberry Pi

To make the system more robust a watchdog is used. The idea is to reboot the system as soon network connection is lost.

* First install the watchdog kernel module with:  
  ‘modprobe bcm2708\_wdog’
* To make the watchdog module loaded automatically at boot time, set in ‘/etc/default/watchdog’:  
  # Start watchdog at boot time? 0 or 1  
  run\_watchdog=1  
  # Load module before starting watchdog  
  watchdog\_module="bcm2708\_wdog"  
  # Specify additional watchdog options here (see manpage).
* Install a watchdog daemon and the chkconfig tool (to activate and deactivate services) with ‘sudo apt-get install watchdog chkconfig’.
* To remove conflicting init scripts the following command must be issued:  
  ‘sudo update-rc.d -f wd\_keepalive remove’.
* Enable the watchdog service with ‘sudo chkconfig watchdog on’.
* Start the watchdog with ‘sudo /etc/init.d/watchdog start’.
* Edit ‘/etc/watchdog.conf’ to set what should be checked by the watchdog daemon. For example to check if the router can be pinged:  
  Unfortunately the ping option does not work, the Raspberry Pi is rebooting continuously. This is because the ping executed by the watchdog daemon does not seem to work and this is a known problem.  
  Therefore we use the setting:  
  file = /home/pi/OwnTools/ownWatchdog  
  #change = 40  
  watchdog-device = /dev/watchdog  
  realtime = yes  
  priority = 1  
  The ‘ownWatchdog.sh’ bash script (see below) runs continuously and checks for the network connection. If all is ok it updates its modification timestamp to let the watchdog know all is ok. If the timestamp is not updated for 40 seconds, the system will reboot.  
  Unfortunately the above does not work, the Raspberry Pi gets in a infinite reboot loop. It seems to do with:  
  *For those who want the gruesome details: wd\_keepalive is supposed to prevent the watchdog (hardware) from timing out during boot. After boot-up watchdog (software) is meant to kill wd\_keepalive and take over. Problems with the startup scripts mean that this isn't happening: wd\_keepalive continues to run and conflicts with watchdog (software), causing the watchdog (hardware) to time-out prematurely. In fact wd\_keepalive isn't needed, because the watchdog (hardware) isn't enabled until watchdog (software) loads. There, clear as mud.*  
  Therefore the ‘change’ is commented out. What does work is that the watchdog is running and when for example the Wifi dongle is unplugged and plugged in again, the system reboots and thus recovers. In combination with the ‘ownWatchdog script’ which reboots when ping fails, we have a robust failure recovery.

## Creating a custom watchdog bash script ‘ownWatchdog’ on the Raspberry Pi to check the network connection

A custom watchdog in the form of a bash script is created. This script checks whether the router (192.168.1.254) or the Samsung Wifi hotspot (192.168.43.1, this is hardcoded in Android) can be pinged. If both cannot be pinged, it does not update its modification timestamp. This will be detected by the watchdog which will then reboot the system in about 15 seconds.  
  
#!/bin/sh

# Start with a touch of this script to feed the watchdog daemon a.s.a.p.

touch -m $0

while [ 1 ]

do

# Start with a sleep to give network time to come up.

echo going to sleep

sleep 10

# Check for router and Android Wifi hotspot

if [ "$(ping -c 1 -W 2 192.168.1.254 | grep '100% packet loss')" -a "$(ping -c 1 -W 2 192.168.43.1 | grep '100% packet loss')" ]

then

echo ping not ok, going to restart network

sudo shutdown -r now

#sudo ifconfig wlan0 down && sudo ifconfig wlan0 up

#sudo /etc/init.d/networking restart

#sudo dhclient wlan0

else

echo ping ok

# Touch this script to feed the watchdog daemon.

touch -m $0

fi

done  
  
To have this script called after reboot add to the end of /etc/rc.local:  
‘/home/pi/OwnTools/ownWatchdog &’.

‘ps –ef’ can be used to check that the watchdog process is running.

# Arduino Uno configuration

# Measurements

The measurements below were done using a digital Multimeter Dynatec 5010C.

The current measurements wer done in the 10A current mode, and the terminals in series with the battery wires.

|  |  |  |
| --- | --- | --- |
| Voltage measurements with 5xAA batteries, batteries appr. 80% charged | | |
| Measurement | **Value** | **Remark** |
| Voltage before regulator, no current | 6.55 V | Measured at battery terminal |
| Voltage before regulator, current 320 mA | 6.30 V (6.26V) | Measured at battery terminal (measured at regulator) |
| Voltage before regulator at full current; 4 motors at max. speed | 5.95 V | Measured at regulator |
| Voltage after regulator, current 320 mA | 4.97 – 4.99 V | Measured at regulator |
| Voltage after regulator at full current; 4 motors at max. speed | 4.96 – 4.98 V | Measured at regulator |

|  |  |  |
| --- | --- | --- |
| Current measurements with 5xAA batteries, batteries appr. 80% charged | | |
| Measurement | **Value** | **Remark** |
| Raspberry Pi only, no Wifi dongle, no camera | 220 mA |  |
| Raspberry Pi only + Wifi dongle, no camera | 320 mA |  |
| Raspberry Pi only + Wifi dongle + camera | 450 - 500 mA | Current varies |
| + Arduino + motor shield | + 40 mA | Delta current measured |
| + 4 motors at speed 100 | + 400 mA | Delta current measured |
| + 4 motors at full speed 255 | + 780 mA | Delta current measured |
| + 4 motors at full speed 255 and all blocking | + 2180 mA | Delta current measured |
| Total current with Raspberry Pi + Wifi dongle, no camera | Appr. 360 mA | Rest mode |
| Total current with Raspberry Pi + Wifi dongle + camera + 4 motors at full speed 255 | Appr. 1250 mA | Action mode |
| Total max. current with Raspberry Pi + Wifi dongle + camera + 4 motors at full speed 255 and all blocking | Appr. 2500 mA | Max. action mode |

|  |  |  |
| --- | --- | --- |
| Video delay measurements | | |
| Measurement | **Value** | **Remark** |
| No motors and using camera setting: input\_raspicam.so -vf -hf -fps 15 -q 50 -ex sports -x 800 -y 600" -o "output\_http.so -p 44445 -w /opt/mjpg-streamer/www" | Appr. 1 s | Connection to Samsung Galaxy S4 with 3G |
| 4 motors at full speed and using camera setting: input\_raspicam.so -vf -hf -fps 15 -q 50 -ex sports -x 800 -y 600" -o "output\_http.so -p 44445 -w /opt/mjpg-streamer/www" | Appr. 2 s | Connection to Samsung Galaxy S4 with 3G |

# Operate the DFRobot using only 3G or 4G

Would it not be nice to be able to operate the DFRobot using only 3/4G? This makes it possible to operate the DFRobot in a remote area where only 3/4G is available, so virtually all over the world. No Wifi required! We will use a Samsung Galaxy S4 phone with a KPN Mobile 3/4G subscription for this experiment and a MacBook in a home network behind a router.

**The first step:**  
Without Wifi it is still possible to operate the DFRobot with an additional phone close to the DFRobot, using the phone’s Wifi tethering capabilities to connect to the DFRobot. Opening the phone’s browser the DFRobot webpage can be seen at <tethering IP address>:44444, for example 192.168.43.182:44444. This way the DFRobot can be operated from the Samsung phone provided the phone is close to the DFRobot.  
**The problem:**  
So far so good, but how can the DFRobot be accessed from any other location through the standard internet? Because the Samsung phone is connected to 3/4G through a provider like KPN Mobile it has no own IP address; when checking the IP address with <http://www.whatsmyip.org/> it will show a KPN Mobile IP address which is not directly related to the phone. This means the DFRobot cannot be accessed from the internet as such.  
It may be that when IPv6 is rolled out phones will get a unique public IP address which can be reached directly, but this can take a couple of years or more.  
**The solution:**Though the Samsung phone connected to 3/4G cannot be reached from the internet, the phone itself cán reach the internet! So the phone could first make a connection to the MacBook. Then a ‘tunnel’ is created which is used to let the MacBook access the DFrobot webserver through a standard browser. This can be accomplished by creating a so-called ‘SSH reverse tunnel’. It is called ‘reverse’ because the DFRobot phone has to initiate it but the tunnel itself is used in the other direction: from the MacBook running a browser and sending requests to the webserver running on the DFRobot. What happens is that first the Samsung phone sets up a standard SSH connection to the MacBook. Then an extra service is started to forward all traffic on port X from the MacBook back to port Y on the DFRobot phone. This is the ‘reversed’ tunnel. This technique is commonly used, for example to bypass firewalls.  
**Steps to set this up:**

* The Raspberry Pi must be able to connect to the Wifi hotspot of the Samsung phone. For this the Wifi credentials of the hotspot connection must be added to ‘/etc/wpa\_supplicant/wpa\_supplicant.conf’.
* On the router, assign a static IP address like 192.168.1.55 to the MacBook. This is needed because we need to forward the SSH port to the MacBook.
* On the router open SSH port 22 to the MacBook.
* On the Samsung phone install ‘ConnectBot’ which is an open source SSH client.
* In ConnectBot create an SSH connection to the MacBook.
* In ConnectBot add two port forwards to this connection:
  + Forward 1, type remote: from port 44444 (MacBook) to 192.168.43.182:44444 (Samsung phone)
  + Forward 2, type remote: from port 44445 (MacBook) to 192.168.43.182:44445 (Samsung Phone)

Two forwards are needed because the DFRobot webserver uses port 44444 for the html page and port 44445 for the MJPEG stream.  
In this case 192.168.43.182 is the address issued by the Samsung Wifi hotspot.

* On the Samsung phone activate Wifi hotspot. The Samsung phone will have the IP address 192.168.43.1 (hardcoded in Android).
* Now put the DFRobot and the Samsung phone in a remote area with 3/4G only.
* On the MacBook, open localhost:44444 in the browser and voilà!

**Experience:**The setup described works quite nice using a Samsung Galaxy S4 with a KPN Mobile 3/4G subscription. The DFRobot which is outside in a 3/4G area can be controlled from a MacBook using a standard browser. Controlling the DFRobot is about as fast as with a Wifi setup. The video performance might be less: the 1 second latency of the MJPEG stream became around 3 seconds in a specific situation (inside) but will obviously depend of the quality of the 3/4G connection. Otherwise the 3/4G experience is the same as the Wifi experience!