

**Setting up the Raspberry Pi based field sensor monitor**  
August 2016

# Contents

<b>1</b>	<b>INTRODUCTION</b>	<b>3</b>
1.1	On Your Computer . . . . .	3
1.1.1	Development Environment . . . . .	3
1.2	Additional hardware . . . . .	4
<b>2</b>	<b>SETUP</b>	<b>4</b>
2.1	Raspberry Pi . . . . .	4
2.1.1	Setting up the Pi: The Easy Way . . . . .	5
2.1.2	Setting up the Pi: The Harder Way . . . . .	5
2.2	Controller . . . . .	9
2.3	Logger . . . . .	9
2.3.1	Radio parameters . . . . .	9
2.3.2	Logging parameters . . . . .	10
2.3.3	Powering the base station . . . . .	10
2.3.4	Powering the logger . . . . .	10
2.4	Putting everything together . . . . .	10
2.4.1	Connecting the controller to the Raspberry Pi . . . . .	10

# 1 INTRODUCTION

This document describes how to set up and use the field monitoring system based on a Raspberry Pi base station and Moteino/Arduino based loggers. It provides a description of the hardware used and how to set up specific parameters in software, such as logging frequency.

In this document, the term “controller” refers to the Moteino which is attached to the Raspberry Pi. It receives data from “loggers”, which are the independent Moteinos with photodiodes attached. The Raspberry Pi takes pictures and is turned off and on by the controller. The term “base station” refers to the combination of Raspberry Pi and Controller, because they work together.

The network operates on a star topology, with individual battery powered loggers reporting back to a base station. Loggers read from their photodiode arrays only when they are told to by the controller. The controller tells the loggers when to read from their sensors, and also turns the Raspberry Pi on and off. Turning the Pi off when it’s not needed results in a dramatic power saving. Like the loggers, the controller is a Moteino microcontroller with a radio attached. It’s necessary to set up radio parameters so that the right loggers are talking to the right controller. These parameters are set out and explained in this document.

The Raspberry Pi acts as a data logger and camera operator. At each logging interval (which must be a multiple of 5 minutes) it is woken by the controller. The controller sends any data received from the loggers to the Raspberry Pi. Once this data has been received, the Pi captures and saves an image before automatically shutting down.

## 1.1 On Your Computer

### 1.1.1 Development Environment

For uploading code to the controller and loggers, you can use the Arduino IDE (Integrated Development Environment). This is a cross platform IDE (you can download it for Windows as well as Mac and Linux) which can be found at <https://www.arduino.cc/en/Main/Software>. There is a good overview of how to use the Arduino IDE at <https://www.arduino.cc/en/Guide/Environment>.

When you have Arduino installed, there are a couple of things you need to do before you can upload code. There are some extra bits of code, called libraries, that the field logging system uses. You can download these, but they are included with the files for this project in *field\_monitor/arduino\_libraries*. These need to go in a specific place for the Arduino IDE to find them. For windows, you should have a directory called *My Documents/Arduino/libraries*. Place all of the folders inside *field\_monitor/arduino\_libraries* into this directory. For Linux, they need to go in */sketchbook/libraries*.

You need to restart the Arduino IDE for it to find the libraries.

You also need to make sure you have the right board selected. The Moteino is very similar to the Arduino Uno, so we can use that as our upload target:

- Open Arduino IDE
- Go to Tools - Board
- Select “Arduino Uno”

You also need to select the right port:

- Open Arduino IDE
- Go to Tools - Port
- Select the port that your Moteino is on. If you’re not sure, unplug the Moteino, and see which one disappears

## 1.2 Additional hardware

You will need an SD card reader and a computer that either has an SD card port or a USB adapter (Google “usb sd card adapter”).

To connect the Moteino to your computer so that you can program it, you will need an FTDI cable like this one <http://uk.rs-online.com/web/p/interface-development-kits/0429307/>. The cable **must** have 5V power and 3.3V logic. The Moteino is designed to run at 3.3V, so 5V logic is too high (the power input can be 5V, because this goes through a regulator which lowers it to 3.3V). This connects to the 6-pin header on the Moteino. Make sure that when you plug it in, the black wire (ground) on the cable connects to the *GND* pin of the header.

## 2 SETUP

### 2.1 Raspberry Pi

The SD card image provided has everything already set up, but there are some more details here if you need to change anything. If you just want to set up a new Pi with the provided image, go to **Setting up the Pi: The Easy Way**.

The Raspberry Pi is set up so that it automatically runs a script when it starts. This script listens for any input from the controller and then captures and saves an image using the camera. Then it shuts down the pi. This script is located in */home/pi/scripts* and it runs from */etc/rc.local*. Scripts run from *rc.local* run as the “root” user (similar to Administrators in Windows). You can look at this file by typing *cat /etc/rc.local*. The Raspberry Pi is turned off and off by the controller; it turns the Pi on, waits for it to complete its tasks, then turns it off.

### 2.1.1 Setting up the Pi: The Easy Way

All you need to set up a Raspberry Pi this way is the Pi itself, an SD card (preferably class 10, 8GB or greater) and card reader, and the camera module.

There is a file called *pi\_logger.img* included in *field\_monitor/raspberry\_pi*. This is an image file, and it's basically an "image" of a working Pi running our code and with our settings. If we flash this image to a new SD card, it will have everything we need for the Pi to work with no setup.

In Linux, use the *dd* tool to flash the new SD card. The command is *dd if=field\_monitor/raspberry\_pi/pi\_logger.iso of=/path/to/new/sdcard*.

In Windows, follow the instruction from the Raspberry Pi Foundation at <https://www.raspberrypi.org/documentation/installation/installing-images/windows.md>. It's not as hard as it looks! You just need to install a program called Win32DiskImager, which will allow you to write to the SD card.

If you use an SD card with a capacity greater than 8GB, you will still only have 8GB of storage unless you expand the partition so that the Pi can access the extra room. To do this, you will need to access the Pi over Ethernet. The image file is already set up to allow you to do this, but you will need to do some setting up on your own computer as well. If you're using Linux, there are instructions in section 2.1.2. If you are using Windows, here's what you need to do (for Windows 7):

- Download and install Putty from <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>
- 

Once you have the SD card flashed, install it in the Raspberry Pi, and connect the camera. There are instructions on how to connect the camera at <https://www.raspberrypi.org/documentation/usage/camera/> (just the "Connecting the Camera" part - the rest is already set up on the SD card).

### 2.1.2 Setting up the Pi: The Harder Way

If you're familiar with Linux, the command line and SSH, you can set everything up from scratch. We will set up the Pi to connect via Ethernet to your machine, so you don't even need an internet connection (this is also useful for downloading data from the Pi when it's in the field - see later section). This section assumes that you already have a Pi running Raspbian (Jessie or later) which you want to set up for use with this system, and that you have root access.

The first thing to do is to set up the Ethernet connection so that your machine can communicate directly to the Pi. If your Pi is already connected to the internet, this isn't really necessary, but it does provide another avenue of data collection when the Pi is located remotely. To connect to the Pi this way, your machine and the Pi need to be within the same subnet. The easiest way to do this is to set up static IP addresses on the Pi and on your machine.

To set a static IP on the Pi, you need to mount the SD card in your machine and edit `/etc/dhcpd.conf`. Insert the following at the bottom of the file (you can replace the IP address with whatever you prefer):

```
interface eth0

static ip_address=192.168.1.10/24
static routers=192.168.1.1
static domain_name_servers=192.168.1.1
```

Now, set up a new ethernet connection on your machine with an IP address within the same subnet. In Ubuntu, this is done as follows (assuming the Pi has the IP address given above):

- Click on the network icon in the menu bar
- Go to *Edit Connections*
- Click on *Add*
- Select the “Ethernet” option
- Give the connection a useful name, like “Pi connect”
- Go to *IPv4 settings*
- Change the *Method* to “Manual”
- Add an address - *Address:* 192.168.1.100, *Netmask:* 255.255.255.0, *Gateway:* 192.168.1.1

You can of course alter these to suit your requirements.

Now you should be able to plug in an Ethernet cable between the Pi and your machine, and use SSH to access the Pi on the IP address you provided. In this case, that would be 192.168.1.10.

Once you are logged in, check that the camera is enabled. Type:

```
sudo raspi-config
```

Select the “camera” option. Select “finish” to set the option and reboot. It’s also a good idea to change from the default password if you haven’t already - use the *passwd* command.

Next, we need to set up the required Python packages. If you have an internet connection, you can just install these via *apt-get* - the following applies if you only have a direct connection to the Pi, via SD card or Ethernet.

There are seven packages located in *field\_monitor/raspberry-pi/packages*. To get them onto the Pi, you can just mount the SD card and copy them into a suitable directory to install from (such as `/usr/local`), or you can copy them over the Ethernet connection using Secure Copy (SCP). To use SCP, the command is:

```
scp path/to/file/being/copied pi@192.168.1.10:/home/pi
```

The packages need to be installed in the right order, as some depend on others. The command to install the *.deb* packages is:

```
sudo dpkg -i <package name>
```

For the others, navigate into the package directory, e.g.:

```
cd netifaces-0.10.4
```

Then install using Python:

```
sudo python setup.py install
```

The installation order is:

- libpython2.7-dev
- libpython-dev
- python2.7-dev
- python-dev
- netifaces
- picamera
- piserail

Once all of the packages are installed, you need to upload a Python script to the Raspberry Pi. The script is located at *field\_monitor/raspberry-pi/process\_data.py* and controls everything relevant to the Pi's field monitoring. As with the packages, you can upload this by mounting the SD card or by using SCP. It's recommended to place it in a directory called */home/pi/scripts*. You also need to create a directory called */home/pi/data* for the CSV files to be stored, and another called */home/pi/data/images* for the images captured by the camera. If you have a different setup, change the *data\_file\_name* and *image\_file\_name* variables in *process\_data.py* to the paths that you want to use.

The *process\_data.py* script needs execute permissions, so make sure to run:

```
chmod +x process_data.py
```

We need the Python script to run at boot, so that when the controller turns on the Pi, the Pi automatically picks up data from the controller, captures an image, and shuts itself down. We can do this by running *process\_data.py* from */etc/rc.local*. Add the following line to */etc/rc.local* before *exit 0* (remember to change the file path, if it's not in */home/pi/scripts*):

```
python /home/pi/scripts/process_data.py &
```

Next there are a couple of things we need to do to allow the Pi to communicate with the controller over serial GPIO. First, edit `/boot/cmdline.txt` and remove the following:

```
console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
```

This will prevent the Pi from sending data over serial when it's booting. Next, we need to disable the console on the serial port. While logged in to the Raspberry Pi, type:

```
sudo systemctl disable serial-getty@ttyAMA0.service
```

Finally, even though we have disabled serial output at boot, there is still a line that prints when the kernel is being uncompressed. To get rid of this, we can get Raspbian to boot from an uncompressed image. This is a little tricky, and you can find the original instructions here <http://raspberrypi.stackexchange.com/questions/24583/ttyama0-disabled-but-still-shows-one-boot-message>. Otherwise, you can follow these instructions:

- SSH into the Pi
- Type the following command (**Note:** If you're using an older Raspberry Pi (less than Model 2), replace `kernel7` with `kernel`):

```
od -A d -t x1 /boot/kernel7.img | grep '1f 8b 08 00'
```

- You should see a line that looks like this (example 1):

```
0017360 1f 8b 08 00 00 00 00 00 02 03 e4 fd 0b 7c 54 d5
```

Or like this (example 2). Notice the extra **08** at the end:

```
0017360 00 00 00 00 00 00 00 00 00 1f 8b 08 00 00 00 00
```

- If your output looks like example 1, the **offset** is 17360
- If your output looks like example 2, the **offset** is 17360 + 08, or 17368
- Type:

```
sudo -i
```

To get an interactive root shell

- Type the following command to create an uncompressed copy of the kernel, replacing the word `OFFSET` with the **offset** number you found above:

```
dd if=/boot/kernel.img skip=1 bs=OFFSET | gzip -d >/boot/kernel_uncompressed.img
```



- Now run the following commands to back up the current compressed kernel, and replace it with the uncompressed version:

```
cd /boot
cp kernel.img kernel_compressed.img
cp kernel_uncompressed.img kernel.img
reboot
```

- If your Pi boots successfully, it has worked

That's it! Your Pi should now be able to run everything it needs for the field monitor.

## 2.2 Controller

## 2.3 Logger

The logger is the Moteino with the light sensor array attached. It runs from 2 x AA batteries and has a low power radio to send data back to the controller. The controller that the logger is sending data back to must have the same *NETWORK\_ID* as the logger, or they can't communicate.

When setting up a new logger, there are a few things that you need to change, and a few things that you can change if you want to. As a minimum, you need to:

- Check that the logger is on the right network, so that it can communicate with the right controller
- Check that the logger's *LOGGER\_ID* is unique on it's network
- Check that the logging frequency is correct

### 2.3.1 Radio parameters

**LOGGER\_ID** The *LOGGER\_ID* is the unique number of the logger. All loggers with the same *NETWORK\_ID* need to have different *LOGGER\_ID*'s. This is partly so that when you look at the resulting data, you will know what logger it came from. It also allows the radios to identify each other, and to prevent clashes between loggers.

The logger ID is transmitted to the controller along with the data from the logger. Because there is a limit to how much data can be transmitted at once, there is a limit to how big the *LOGGER\_ID* can be, because it is only allowed to take up one byte, or eight bits. The highest number you can represent with eight bits is 255. So, including zero, we can have up to 256 unique *LOGGER\_ID*'s on one network (networks work the same way). One of these numbers (the number 1) is actually taken by the controller, and the number 255 is used as a "broadcast", meaning that all loggers on the network can see data transmitted to 255.

So, when you are setting up a new logger, set the logger ID to something greater than 1 and less than 255, and make sure that no other loggers with the same *NETWORK\_ID* have the same *LOGGER\_ID*.

**NETWORK\_ID** The *NETWORK\_ID* is used to identify which loggers are on the same network, i.e. which loggers can talk to each other. If you have a Raspberry Pi set up in one field, with a controller with a *NETWORK\_ID* of 100, all of the loggers that need to talk to that controller must also have a *NETWORK\_ID* of 100. If you set up another base station in a different field nearby, you probably don't want the loggers in the first field to also be transmitting to that controller.

So, each controller has a unique *NETWORK\_ID* (like *LOGGER\_ID*'s, these must be greater than 1 and less than 255), and all loggers that send data to that controller have the same *NETWORK\_ID*.

**GATEWAY\_ID** The *GATEWAY\_ID* is used to identify the controller, which acts as a "gateway" to the Raspberry Pi. So, it's the *LOGGER\_ID* of the controller. Make sure that you put the right *GATEWAY\_ID* for the controller that the logger is sending to.

**FREQUENCY** This is just the frequency of the radio chip - 433MHz is within the Industry, Scientific and Medical frequency band, which means it is a legal frequency for low power radio communications. Don't change this unless you know what you are doing.

**ENCRYPTKEY** This is an encryption key - only loggers with the same key can understand each other's transmissions. For this reason, like the *NETWORK\_ID*, all loggers that talk to each other (including the controller) must have the same *ENCRYPTKEY*. It **must** be 16 characters long. This encryption also means that no one else can pick up packets being sent on your network.

### 2.3.2 Logging parameters

#### NUM\_PHOTODIODES

### 2.3.3 Powering the base station

### 2.3.4 Powering the logger

## 2.4 Putting everything together

### 2.4.1 Connecting the controller to the Raspberry Pi

To enable the controller and the Pi to communicate, you need to connect the TX and RX pins of the Moteino to the TX and RX pins of the Pi as follows:

- Find the pin on the Moteino labelled "TX"

- Locate the “RX” pin of the Pi. If you are looking at the Pi with the USB ports closest to you, this is the fifth pin down from the top of the right hand header
- Connect the “TX” pin of the Moteino to the “RX” pin of the Pi
- Find the pin on the Moteino labelled “RX”
- Locate the “TX” pin of the Pi. If you are looking at the Pi with the USB ports closest to you, this is the fourth pin down from the top of the right hand header
- Connect the “RX” pin of the Moteino to the “TX” pin of the Pi