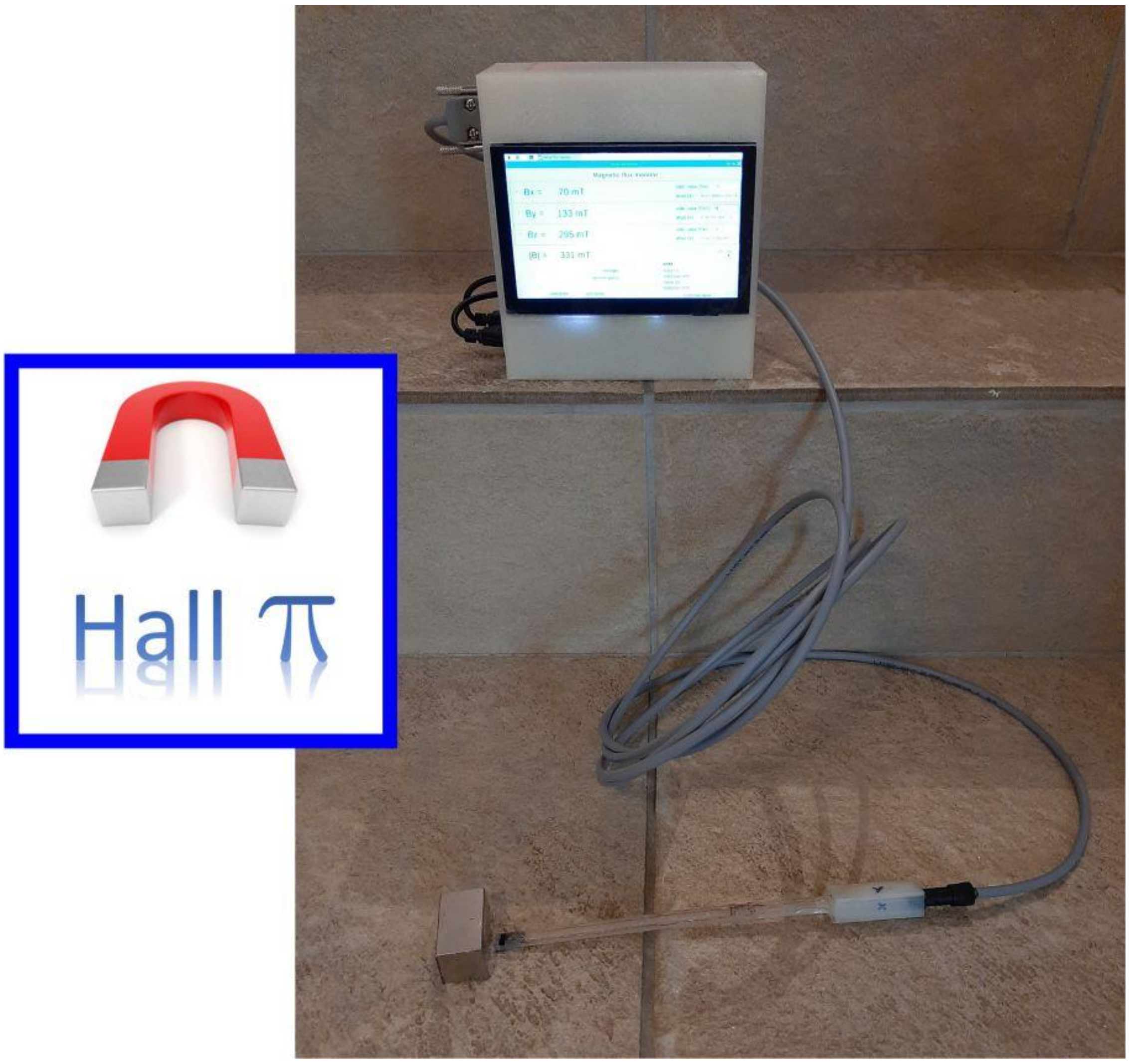


# The Hall-π project

The idea for this little project started when I had to buy probably the 10th Hall-probe for ca. 1000 € after the old one was broken again by one of my students. I did a bit of research on the available sensors and figured that one can buy useful linear Hall-sensors already for 1-2 € / piece. Then I decided to start a little side-project to build a Gaussmeter run from a Raspberry Pi Mini-Computer. I had sufficient time to finalize it during the Corvid-19 lockdown in spring 2020.

This report is on setting this all up for a 3-axis Hall-probe, but you can use the same hardware and programs together with other sensors (e.g. temperature, pressure, etc.) as long as they are operated by 5V and the measured effect is represented by a voltage.

Initially I set myself the following aims: The final device should:

- measure magnetic flux (at least 2 T) with high precision in 3 dimensions;
- have calibration options, that can be saved for several different sensors;
- run like a multimeter (touchscreen, battery);
- somehow transfer its data to PCs using a standard protocol;
- cost all in all less than 200 €

I had fun doing this little project, learned a lot about Raspberry Pis and Python. However, my coding qualities in Python are not great, so if you find ways of optimizing my code please let me know, so that I can update the file or this report.

Have fun and stay healthy, yours Peter Blümler (April 2020)

## What you need: (but better read the instruction before ordering)

To realize this using the following parts (prices from March 2020)

- Raspberry Pi B incl. power supply and SD-card
  https://www.raspberrypi.org/ ca. 70 €
- 5“ HDMI touchscreen LCD-monitor - 800 x 480 HD
  (many monitors are suited, but they need the touch-function be connected via USB and **not** through the GPIO-port of the Raspberry!!!) ca. 40 €
- Waveshare Raspberry Pi AD/DA expansion board with
  ADS1256 DAC8552 Sensor Interface
  https://www.waveshare.com/wiki/High-Precision_AD/DA_Board ca. 30 €

  You could also use another ADC (like the MCP3008), but I wanted a higher digital resolution than 10bit. The Waveshare offers 24 bit.
- 3 Hallsensors (e.g. CYSJ302C from Sonnecy)
  https://www.sonnecy-shop.com/en/hall-elements-ics/linear-hall-sensor-elements.html ca. 6 €

Additionally the following things will be needed/useful:

- USB keyboard (for programming the Raspberry before switching to touchscreen mode) eventually a HDMI/VGA or HDMI/DVI-adapter if your PC-monitor has no HDMI input
- USB mouse
- Solder iron
- 3D printer (for making nice probe supports and tailored housing
- A 5V powerbank or a 5V battery to run the device cordless. Check the voltage and the power (2.5 A!) before buying. I used a 10 Ah mini powerbank from Jonkuu with 2 * 2.4 A and 5V.
- Some Nylon or brass Hex spacers
- USB A, USB B micro and HDMI connectors for soldering to compact the device

This instruction is subdivided in the following sections:

1) The Hall-sensor: Building a 3-axis Hall-probe

2) Setting up the Raspberry Pi hardware

3) Installing the software

4) Establishing contact to a PC

5) The Hallpi-GUI

6) Setting up the touch screen

7) Polish and finish

8) Calibration

Possible Improvements

Before you start, please download the `Hallpi.zip` file which is an archive with three directories:

- `3D print` : contains STL files for 3D-printers
- `copy to raspberry` : this you should copy to the Raspberry-Pi using a pen drive
- `manuals datasheets` : some PDFs about ADDA card, sensors, etc.

It might be advisable to do section 1 in parallel with the rest, because building and soldering the probe takes much longer than installing the software, and you might need a functioning Raspberry Pi with the Hall-pi GUI to test your sensors.

## 1) The Hall-sensor: Building a 3-axis Hall-probe

I have tried 3 different sensors:

1) CYSJ362A from Sonnecy https://www.sonnecy-shop.com/en/hall-elements-ics/linear-hall-sensor-elements.html
This is a linear GaAs-Hall Effect Sensor 3-4 mV/mT, which is quite sensitive but probably only if you run it with Vcc = ±12V in (+12V on pin 1 and -12V on pin 3). With +5V only on pin 1 I expect more like 1.4 mV/mT with a linearity of 2%. But its max. flux of 3T is nice together with a price of 1.2 €. This one I used for this project.

2) CYSJ302C from the same company. This has a sensitivity of ca. 0.6 mV/mT at 5V and a maximum flux of 2T, I used this mainly for comparison.

3) A more professional device the HE244 from Advanced Sensor Technology. This one goes up to 10 T with 0.2 mV/mT. I only bought one for high field checks, because it costs 135 €.

However, this choice was predominantly due to my “professional hobby” of building permanent magnets. So 3T is perfect. If you want to go to other fields with higher sensitivity, there is a lot on the market. A variety of other linear Hall-sensors is given here http://www.hallsensors.de/Hall-IC.htm

Vendors of commercial Hall-probes usually avoid non-magnetic metals as protective housings, because if you want to measure AC magnetic fields or move the sensor quickly through it, induced eddy currents might interfere with your measurement. However, if you stick with static measurements in static fields an aluminium cover is fine and probably extends the lifetime of your sensors.

The next step is to attach 3 Hall-sensors in orthogonal directions. You might to consider to get this part made from a professional workshop or if you have access to a CNC-mill you might mill a suitable support structure from some polymer or metal. I have a nice 3D-printer, hence I printed an entire probe structure from a nylon filament.
The STL file (`3axis Hall probe.stl` ) for this piece is available in the `Hallpi.zip` archive in the folder 3D print. On my Ultimaker 3 a 100% filled print with ultimate resolution took about 4 h, but the device is rather rigid (see Fig. 1).

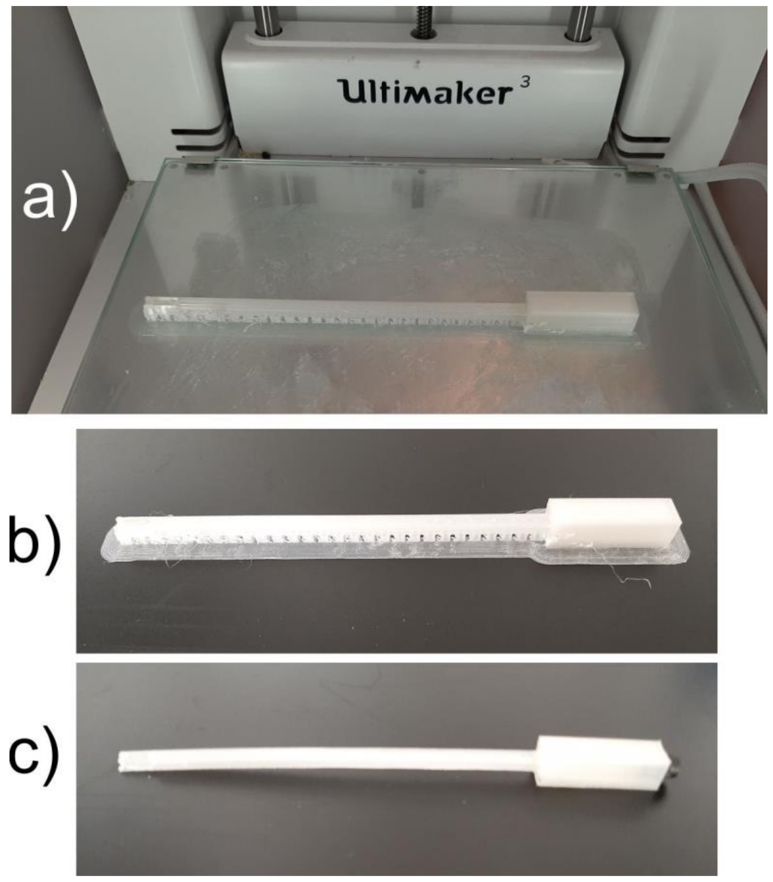


Fig. 1: a) Finished print on the 3D-printer, b) Print with supporting structures, which were removed in c)

The probe design has small orthogonal holes for the Hall-sensors (x,y,z in Fig. 2 insert) where the sensors can be glued to. There are also grooves for the wires to contact them along the shaft. These grooves should have feedthroughs into the hollow grip, but in my printout I had to drill them with a 2 mm drill .

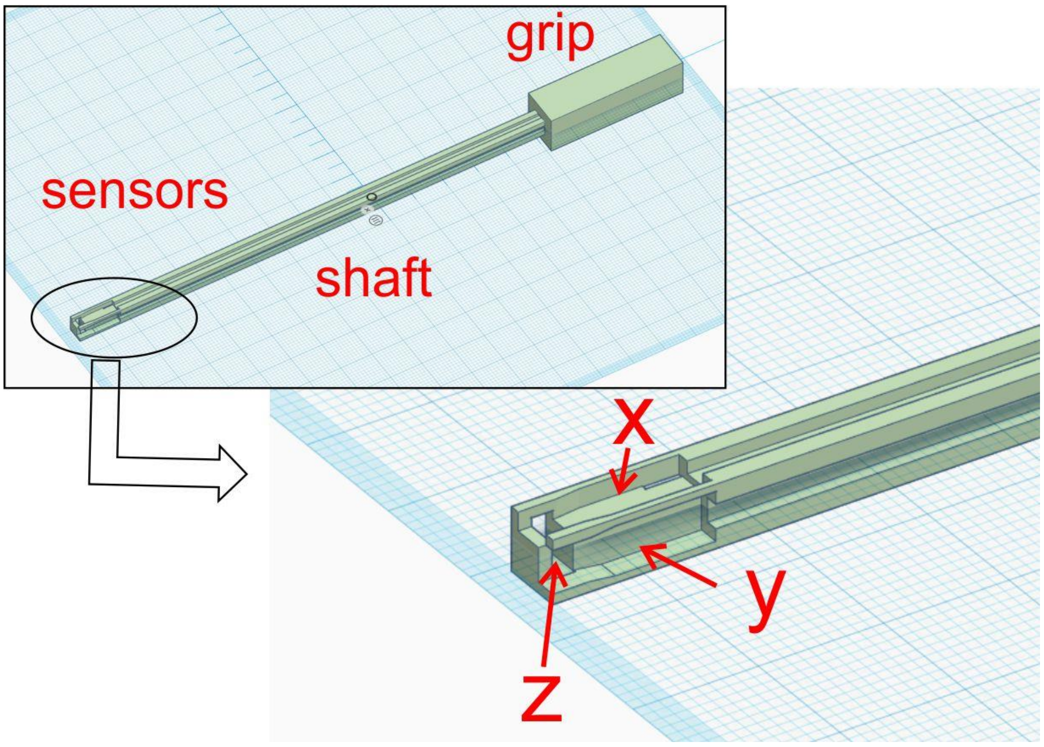


Fig. 2: Schematic view of the probe which has three sections. At its tip there are three orthogonal holes for each sensor followed by a long groove for the connecting wires along the shaft. In the hollow grip, the wires can be soldered to a connector.

The hole in the shaft is for Binder-connectors, I had chosen because I had a lot inherited from some lab clearing (Binder snap in connectors Series 719 or IP40 Subminiature, https://www.binder-connector.com/en/products/subminiature-circular-connectors#filter:contacts=_5&lock=Snap%20In). You can of course use your favorite connector. I like to document my pin assignment as a reference for the future… you do not have to follow that.
Solder thin varnished copper wires to each sensor as shown in Fig. 10a. They should be ca. 5 cm longer than the entire probe. You can encode your wires (5V, VH and GND) either by length (5V and GND of the 3 sensors can be soldered together on the shaft) or by color (painting the wire), cf. Fig. 3a. I used some varnish (e.g. nail polish) to put a thin insulation layer over the solder spots at the sensors to avoid shorts. After that has dried I twisted the wires and tested if the sensors worked by connecting them through a breadboard to the AD/DA card and the Raspberry (so you have to complete section 2-5 of this manual first). If you bring a small permanent magnet to the sensors the flux display should change dramatically (of course this depends on your settings of the offset and calibration, see section 5).
If they all work, glue the sensors in the sockets at the tip of the probe and the twisted wire into the grooves. I have used 5 min epoxy for that.

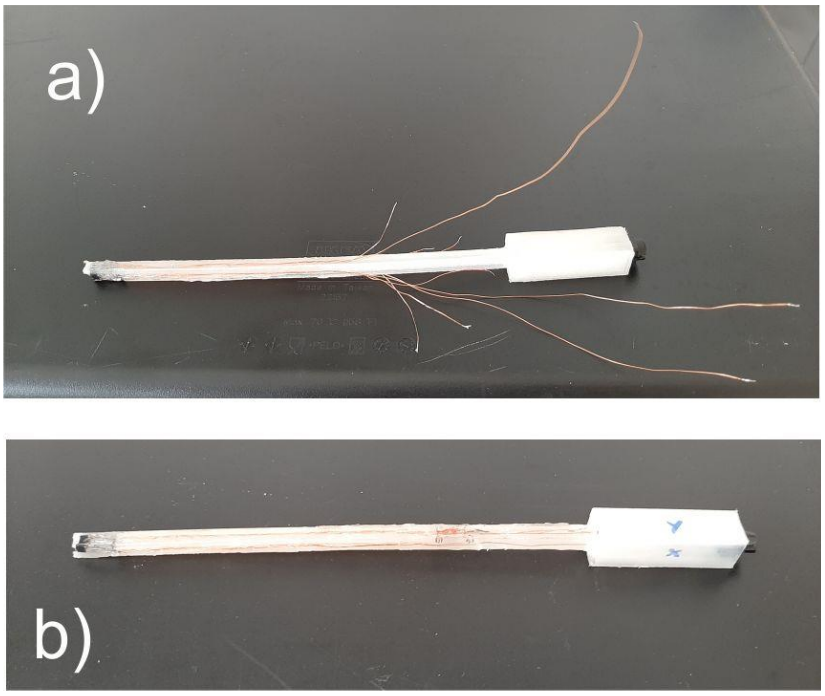


Fig. 3: a) Wires mounted to the probe, b) finished probe .

Finally I want to show the connections from the probe via a cable to the Raspberry (Fig. 4). I do that mainly as a reference for me. But it is advisable to document your colors etc. somewhere as well.

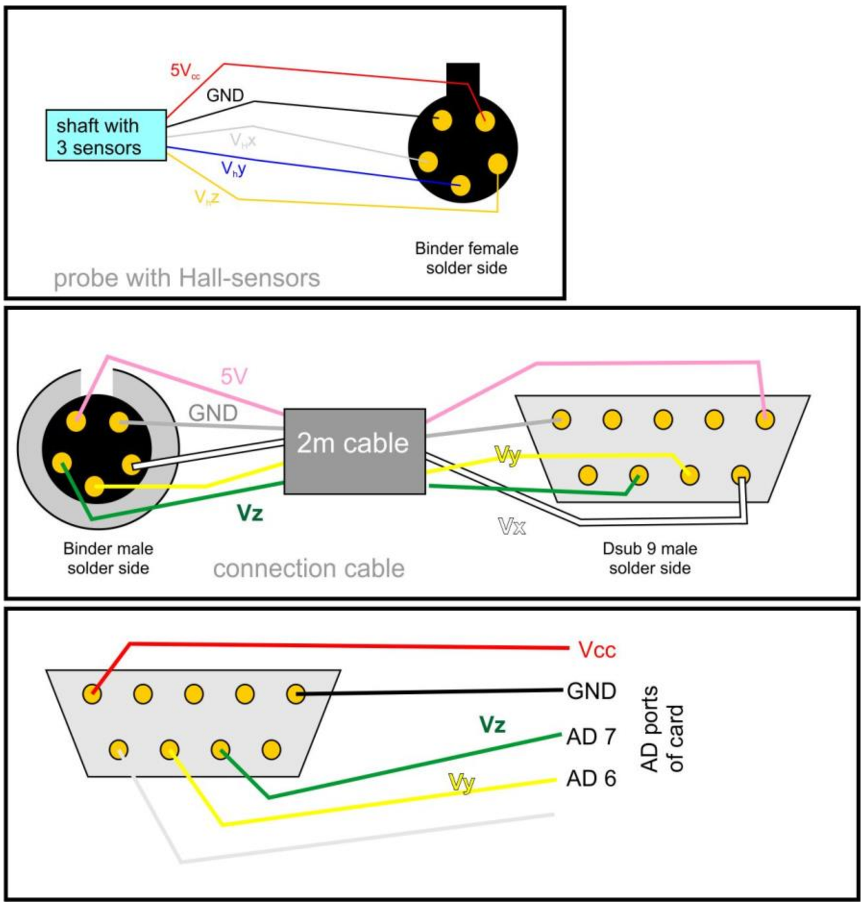


Fig. 4: Color coding and positions on connectors as I have used it considering the things that were just in my possession.Note that all connectors are displayed from the solder side

## 2) Setting up the Raspberry Pi hardware

Figure 5 shows all three major components that we have to buy to assemble the necessary hardware.

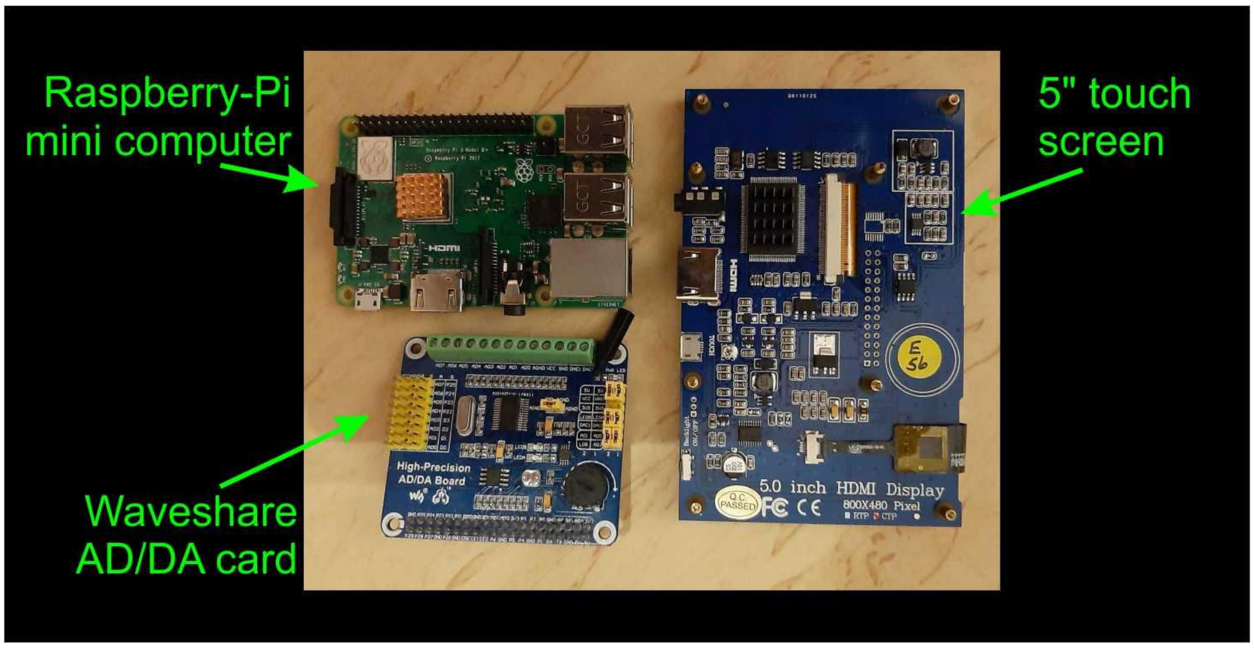


Fig. 5: Commercial parts: Raspberry-Pi computer, touch screen and AD/DA board before assembly.

**Careful with that monitor, Eugene!**
As I already emphasized in the beginning, do not use a monitor, which has its touch functions transferred via the GPIO bus of the Raspberry Pi. Such monitors typically have a lengthy connection bar at their lower side (see for instance this one https://www.rasppishop.de/Resistives-IPS-Touch-Screen-Display-35-fuer-Raspberry, a nice display but not suitable for this project). The video signal can either be transferred via HDMI or the display-port of the Raspberry. The touch function is best connected via USB like in the one shown here.

1) First screw the Raspberry on top of the screen. I used some nylon 2 mm Hex spacers that were not shipped with the monitor to give the Raspberry a bit more room for ventilation. (see Fig. 6)

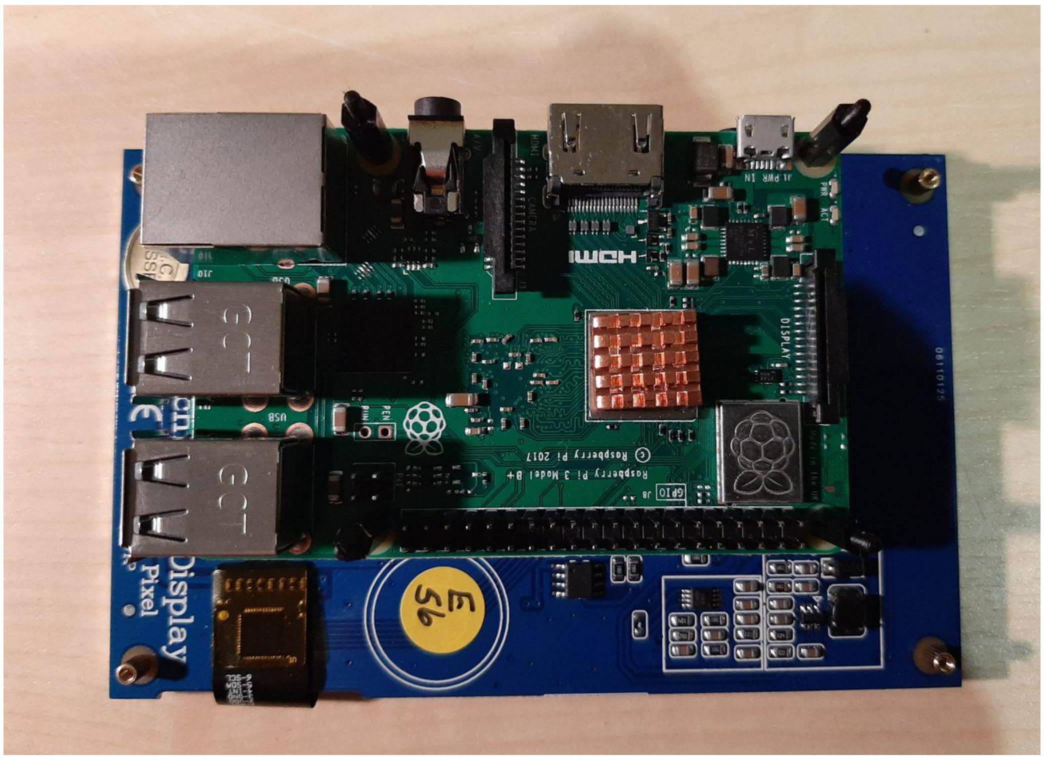


Fig. 6: Raspberry-Pi screwed on the back of the screen using Hex-spacers.

2) Then do the same with the Waveshare AD/DA board, connect its lengthy connector carefully to the GPIO-port of the Raspberry, and use some 15 mm high spacer screws. Should look like Fig. 7.

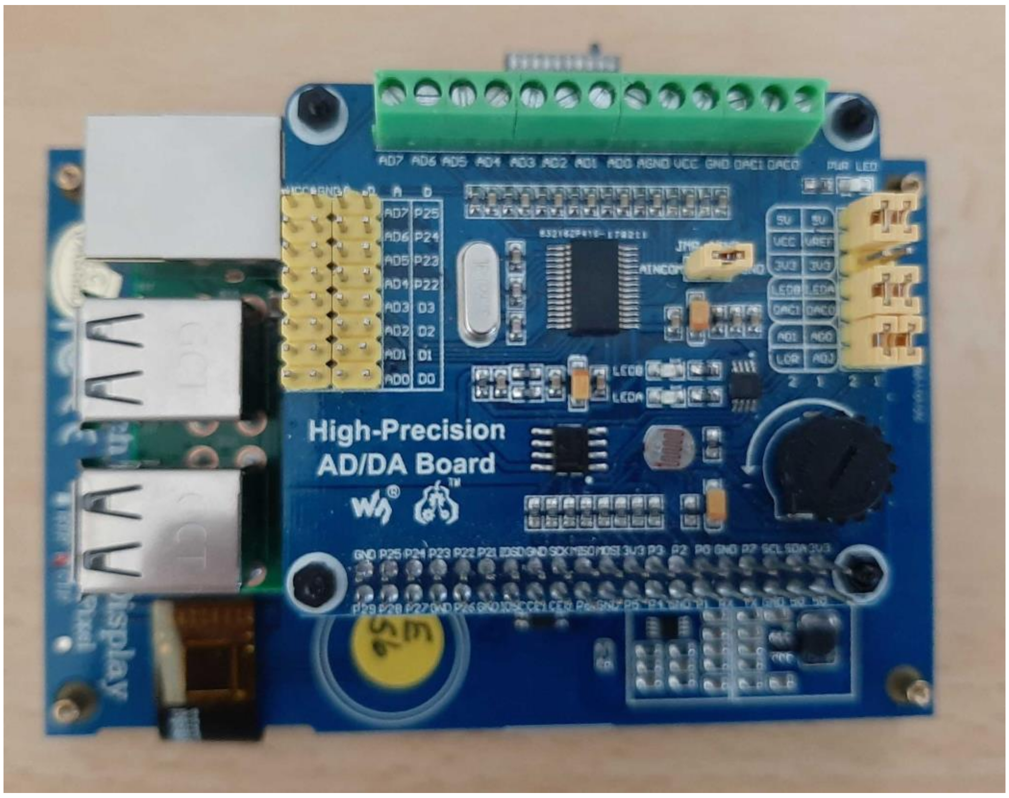


Fig. 7: Finally the Waveshare AD/DA board is screwed on top of the Rapberry after connecting it to the GPIO port (see also Fig. 8).

3) Before we tidily pack up everything in the last step, I recommend to have a first operational setup. Before we install the touchscreen display in chapter 6, I recommend to attach a normal PC-monitor via the HDMI-port, a USB keyboard and an USB mouse. This might look like Fig. 8.

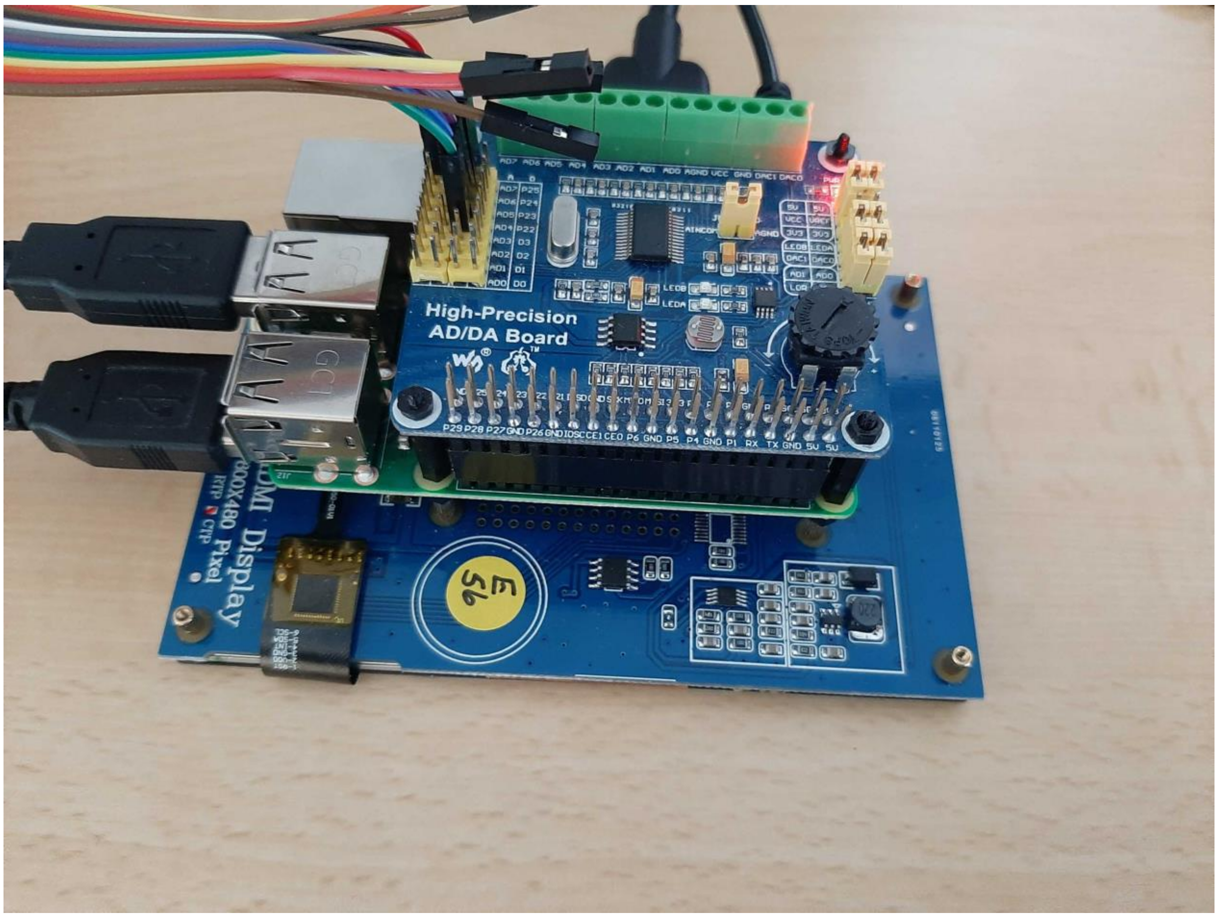


Fig. 8: The hardware assembly is preliminary connected to a PC monitor (HDMI connector in the back), USB mouse and keyboard (on the left). The AD-port can now be connected to a breadboard via jumper cables (top). In this photograph you can also see the GPIO connection between Raspberry and the AD/DA board (black bar in the front).

Here I have attached the sensors by breadboard jumper cables for a first test.
In my setup the operating voltage and ground are already joined inside the sensor, so they do not have to be supplied separately. If you however decide to operate different sensors from the ADC-card, you might to do so.

The 3 Hall sensors we made in chapter 1 are now attached to the ADC on the Waveshare board. They go to the red encircled ports (the 8 ADC input ports, No. 3 in Fig. 9 ).

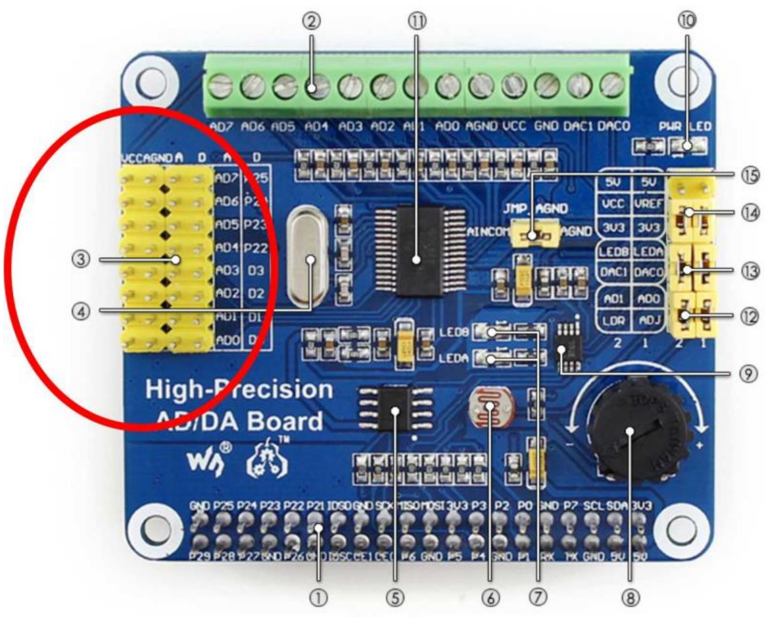


Fig. 9: Waveshare AD/DA board: We only use the 8 encircled AD ports (No. 3).

The three sensors are attached as shown in Fig. 10. The +5V (pin 1) of each sensor are joined either inside the probe or later in a connector to the Raspberry. For the moment I would recommend to do this via a breadboard, which gives most flexibility in this setup phase. Pin 2 is the flux dependent Hall-voltage $V_H$ which we want to measure this goes to the A connector of three different ADC-channels. If you want my GUI-program without making changes, you must connect them like shown in b). (Bx-sensor to AD5, By to AD6 and Bz to AD7). Pin 3 and 4 of each sensor can be soldered together to reduce wiring inside the probe and then they are connected to one of the GND-ports of the ADC. Now let's see if we can read them into Python.

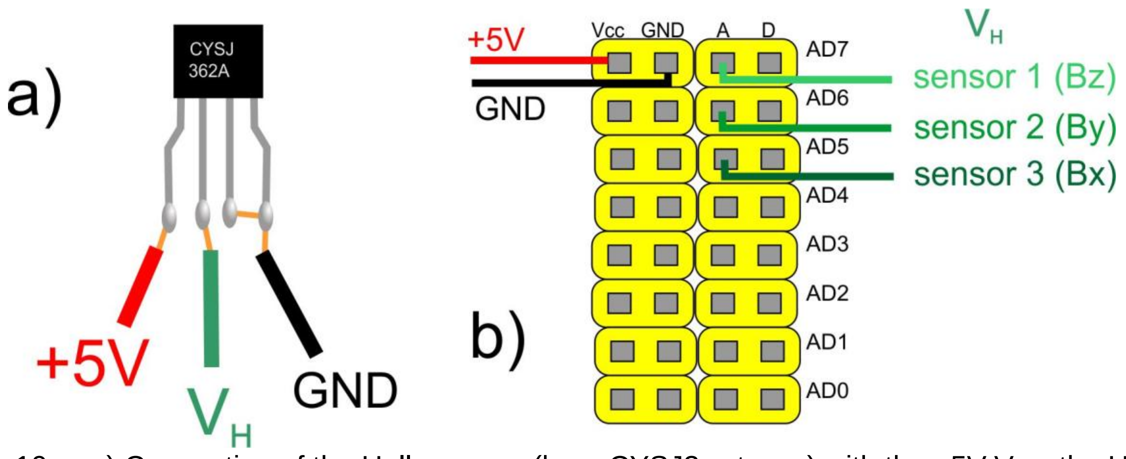


Fig. 10: a) Connection of the Hall-sensors (here CYSJ3xx types) with the +5V Vcc, the Hall-voltage $V_H$ and ground (GND). b) Schematic drawing for connecting 3 such Hall-sensors to the AD-ports of the Waveshare-board.

## 3) Installing the software

a) first install your Raspberry-Pi from the SD-card (or making your own SD-card using https://www.raspberrypi.org/documentation/installation/installing-images/). Ensure you have a internet connection via WiFi or LAN.

b) update the operating system. Open a terminal and type

```
~$ sudo apt-get update
~$ sudo apt-get dist-upgrade
```

c) Configure the SPI-port on the raspberry: open a terminal and type

```
~$ sudo raspi-config
```

then select number 5 Interfacing Options
then select P4 SPI
enable it

d) Go to your home directory and create a directory `hallpi`

```
~$ cd
~$ mkdir hallpi
```

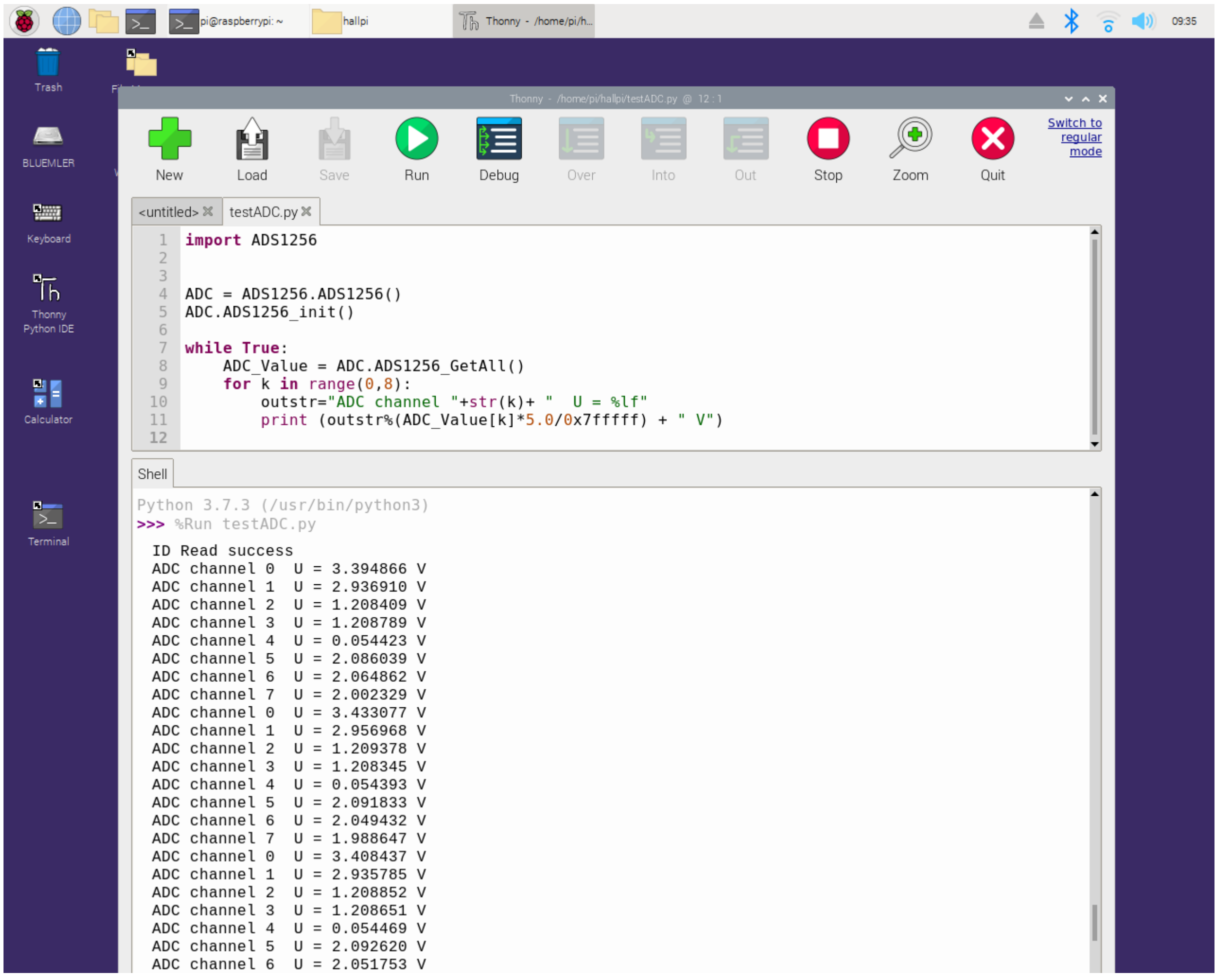
e) now copy at least the following programs (download from this webpage) using a pen-drive or SD-card.

- `ADS1256.py`
- `config.py`
- `testADC.py`
- `Hallpi.py`

Of course you can copy the entire directory `copy to raspberry`
Done!

For an easy check if your hardware works you can run the little program `testADC.py`. Your screen should look like Fig. 11 (running it from Thonny Python):

You can stop it by pressing either STOP in the menu bar of Python or giving a Ctrl+C from the keyboard



```python
import ADS1256


ADC = ADS1256.ADS1256()
ADC.ADS1256_init()

while True:
    ADC_Value = ADC.ADS1256_GetAll()
    for k in range(0,8):
        outstr="ADC channel "+str(k)+ "  U = %lf"
        print (outstr%(ADC_Value[k]*5.0/0x7fffff) + " V")

```

```
Python 3.7.3 (/usr/bin/python3)
>>> %Run testADC.py

 ID Read success
 ADC channel 0  U = 3.394866 V
 ADC channel 1  U = 2.936910 V
 ADC channel 2  U = 1.208409 V
 ADC channel 3  U = 1.208789 V
 ADC channel 4  U = 0.054423 V
 ADC channel 5  U = 2.086039 V
 ADC channel 6  U = 2.064862 V
 ADC channel 7  U = 2.002329 V
 ADC channel 0  U = 3.433077 V
 ADC channel 1  U = 2.956968 V
 ADC channel 2  U = 1.209378 V
 ADC channel 3  U = 1.208345 V
 ADC channel 4  U = 0.054393 V
 ADC channel 5  U = 2.091833 V
 ADC channel 6  U = 2.049432 V
 ADC channel 7  U = 1.988647 V
 ADC channel 0  U = 3.408437 V
 ADC channel 1  U = 2.935785 V
 ADC channel 2  U = 1.208852 V
 ADC channel 3  U = 1.208651 V
 ADC channel 4  U = 0.054469 V
 ADC channel 5  U = 2.092620 V
 ADC channel 6  U = 2.051753 V
```

Fig. 11: Screen dump of the output from `testADC.py`.

You can vary the magnetic flux to see if the channels of the Hall-sensors (here 5,6,7) change their values….they do?! Great! (if not double check that your SPI channel is switched on and the files ADS1256.py and config.py are in the same directory)

So were almost there, but before we start the measurement GUI we now have to install MQTT.

## 4) Establishing contact to a PC

After pondering about the different possibilities on exchanging data between two PCs (e.g. USB, Ethernet, FTP…), I decided to use MQTT (https://de.wikipedia.org/wiki/MQTT) as the transfer protocol. This decision was made mainly because it was best documented for Python and Raspberrys.
A nice introduction is found here: https://kitflix.com/communication-between-raspberry-pi-and-pc-using-mqtt/

## Installing MQTT using mosquito:

### On the Raspberry-Pi:

```
~$ sudo apt-get install -y mosquitto mosquitto-clients
~$ sudo systemctl enable mosquitto.service
~$ sudo pip install paho-mqtt
~$ sudo pip3 install paho-mqtt
```

### On the PC:

```
> pip3 install paho-mqtt
> python -m pip install --upgrade pip
```

Reboot the PC

**Transferring data from Hallpi to the PC:**
(you can try after you have understood and ran the Hallpi GUI in the next chapter)

This is a small example in Python (you can easily install IDLE Python on your PC from https://www.python.org/downloads/) how to receive the data from the Raspberry. You have to insert the IP-address (here 192.168.0.132) of your Raspberry (displayed when the program Hallpi.py is started. This program is in the top directory of the ZIP-file named `test_mqtt_client.py`.

```
# little example how to receive data from the raspberry pi
# send via MQTT over the internet
#
# P. Blümler 2020
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc):
   if rc==0:
        print("connected OK Returned code=",rc)
   else:
        print("Bad connection Returned code=",rc)
   client.subscribe("hallpi")

def on_message(client, userdata, msg):
  print(msg.payload.decode())


client = mqtt.Client()
client.connect("192.168.0.132",1883,60)

client.on_connect = on_connect
client.on_message = on_message

client.loop_forever()
```

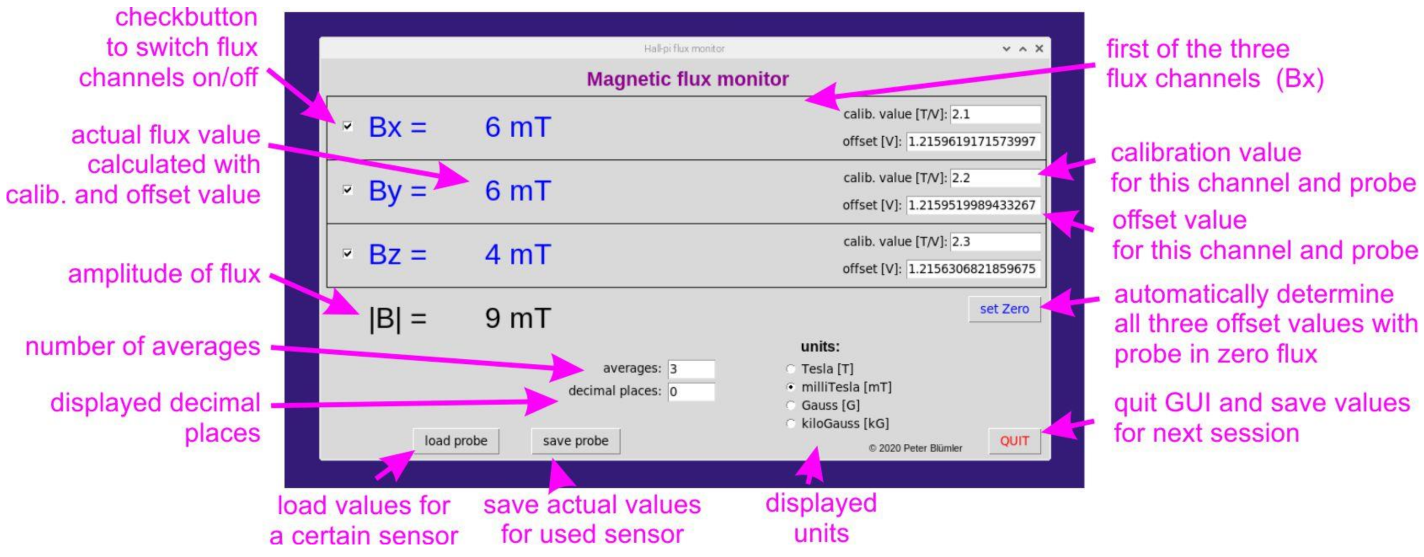
then start the Hallpi.py program on the raspberry and then the PC should display

```
connected OK Returned code= 0
Bx =       0.0380 T
By =       0.0408 T
Bz =       0.0422 T
Bx =       0.0398 T
By =       0.0413 T
Bz =       0.0412 T
Bx =       0.0405 T
…
```

I did this in a very basic way. If you want to alter the data-style transmitted via MQTT just alter the format of pubstr in line 185 (`client.publish("hallpi",pubstr)`; to match your desired format.

## 5) The Hallpi GUI

Now load Hallpi.py into Python and if everything is ok your screen should display the graphical user interface (GUI) of the Hall-pi program (see Fig. 12) after you have pressed RUN.


checkbutton to switch flux channels on/off
actual flux value calculated with calib. and offset value
amplitude of flux
number of averages
displayed decimal places
first of the three flux channels (Bx)
calibration value for this channel and probe
offset value for this channel and probe
automatically determine all three offset values with probe in zero flux
quit GUI and save values for next session
load values for a certain sensor
save actual values for used sensor
displayed units
Hall-pi flux monitor
set Zero
load probe
save probe
QUIT

Magnetic flux monitor

[✓] Bx = 6 mT  calib. value [T/V]: 2.1  offset [V]: 1.2159619171573997

[✓] By = 6 mT  calib. value [T/V]: 2.2  offset [V]: 1.2159519989433267

[✓] Bz = 4 mT  calib. value [T/V]: 2.3  offset [V]: 1.2156306821859675

|B| = 9 mT

averages: 3
decimal places: 0

units:
○ Tesla [T]
● milliTesla [mT]
○ Gauss [G]
○ kiloGauss [kG]

© 2020 Peter Blümler

Fig. 12:  The Hall-pi GUI with explanations.

I hope you find this GUI intuitive. Concerning the code I have to confess that this is my first bigger Python code. I wanted to do it more elegantly but after a few trials I gave up and used whatever worked. Due to the importing of elements I find Python syntax sometimes very confusing and inconsistent.

The GUI shows the three flux components (Bx, By, Bz) in 3 separate boxes. Each of them can be switched on or off by ticking the checkbutton on the left. This is for using single or dual channel probes (with only one or two Hall-probes) with the same device. The checkbutton is followed by a display of the actual value, *B*. This value is calculated by subtracting an offset, $V_{off}$, from the measured ADC-voltage, $V_H$, and is then multiplied by a calibration constant, *A*.

$$B = (V_H - V_{off})\,A$$

Both offset, $V_{off}$, and calibration constant, $A$, can be manually changed for each flux component on the right side. Just type in a value and press RETURN.
Below the three flux channels the amplitude of the flux is displayed in black according to

$$|B| = (B_x^2+B_y^2+B_z^2)^{0.5}$$

The offsets can be automatically determined by the (set Zero) button on the right. Therefore, the sensors should be brought into a shielded calibration chamber (for absolute measurements) or in the background flux (relative measurements). The offsets are then determined by averaging 100 samples.
The values and settings can be saved and loaded for each probe, in case you want to attach more than one. Otherwise the actual values are saved in a file default.csv when leaving the GUI by pressing the QUIT-button in the lower right corner.

There are three further items, the number of averages, i.e. how may individual measurements are averaged. If you increase this number the display will need much more time to change, but the values become more stable.
The number of decimal places behind the decimal point. Finally there is a field for altering the displayed units between useful SI (Tesla) and CGS (Gauss) units.

Of course one doesn't want to start the GUI always through the Python console, so we will make a little desktop icon to execute it. For that open the text editor of the Raspberry (or use nano in the directory /home/pi/Desktop). There you type the following

```
[Desktop Entry]
Name=Hallpi
Icon=/home/pi/hallpi/Hallpi_Logo.xpm
Exec=sudo python3 /home/pi/hallpi/Hallpi.py
Type=Application
Encoding=UTF-8
Terminal=false
```

Save it to the directory `/home/pi/Desktop` as `Hallpi.desktop`.
Then open a terminal and..

```
~$ cd /Desktop
~$ sudo chmod +x Hallpi.desktop
```

Now you can simply double-click on the Hallpi icon and the program will start. To avoid the annoying question about executables, open the `File Manager` of the Raspberry Pi (in the Menu click `Edit-> Preferences -> General`) and tick the box "Don't ask options on launch executable file" on the "behavior")

## 6) Setting up the touch screen

If all functions work to your satisfaction, you can switch the display to the touch screen. Depending on the model that might be slightly different. The one I used needs to be connected via an HDMI-cable to the Raspberry (video) and an USB-cable (touch function).

The software installation has to be matched to the used device as well. Here is a good overview: http://www.lcdwiki.com/How_to_install_the_LCD_driver.

In my case that was

```
~$ sudo rm -rf LCD-show
~$ git clone https://github.com/goodtft/LCD-show.git
~$ chmod -R 755 LCD-show
~$ cd LCD-show
~$ sudo ./MPI5001-show
```

If you need to rotate the screen:

```
~$ cd LCD-show
~$ sudo ./rotate.sh 90
```

If you like to switch back to a normal PC-screen

```
~$ cd LCD-show
~$ sudo ./LCD-hdmi
```

You might also want to have a keyboard on the touch screen (doesn't come automatically). Different versions are available which are all not great, but Matchbox worked best for me. You can install it by

```
~$ sudo apt-get install matchbox-keyboard
```

…and reboot…DONE!

## 7) Polish and finish

Finally, we want to pack everything in a nice housing. You can find a straight-forward 3D printable case in the STL-file `3D print\Hallpi case.stl`. The result is shown in Fig. 13. (It can be closed with a cover, which can be printed from the file `3D print\Hallpi Cover.stl` ). I used PLA for this which took ca. 24 h). After cutting/filing the support structures you can try to insert the instrument (Screen + Raspberry Pi + Waveshare AD/DA board as in Fig. 7), but be very careful that you do not damage anything (e.g. the SD card at the edge of the Raspberry). Of course you have to strip it from all cables first.

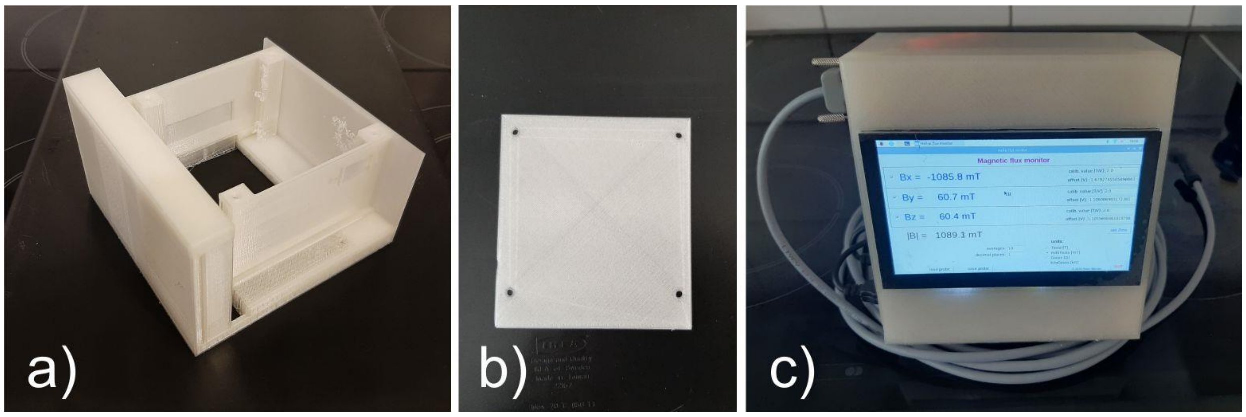
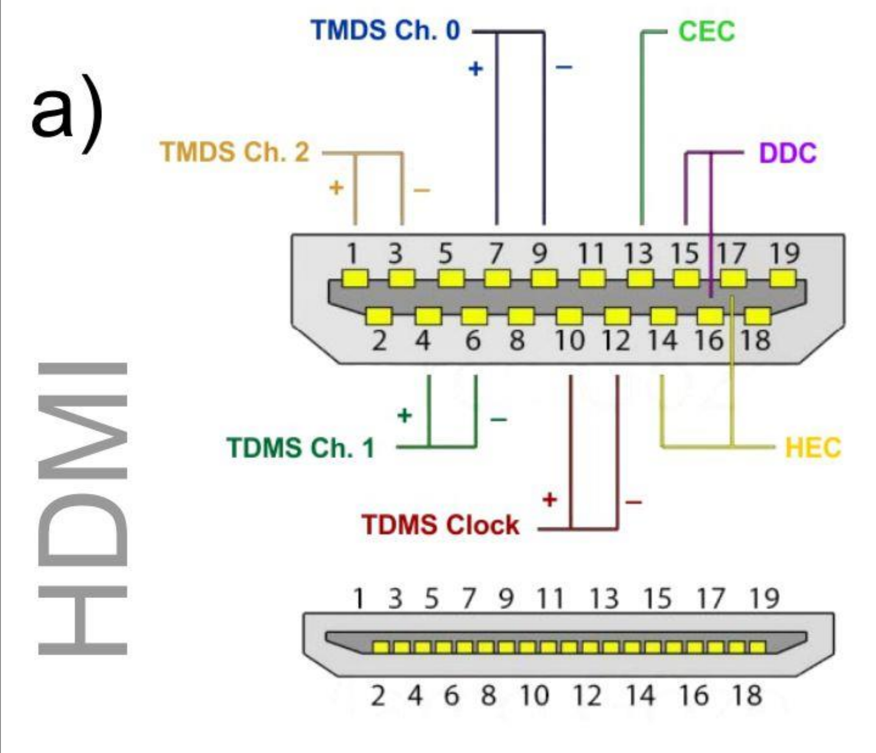


Fig. 13: a) 3D printed case for screen, raspi, waveboard, connectors and a powerbank, b) cover for the back, c) assembled device.

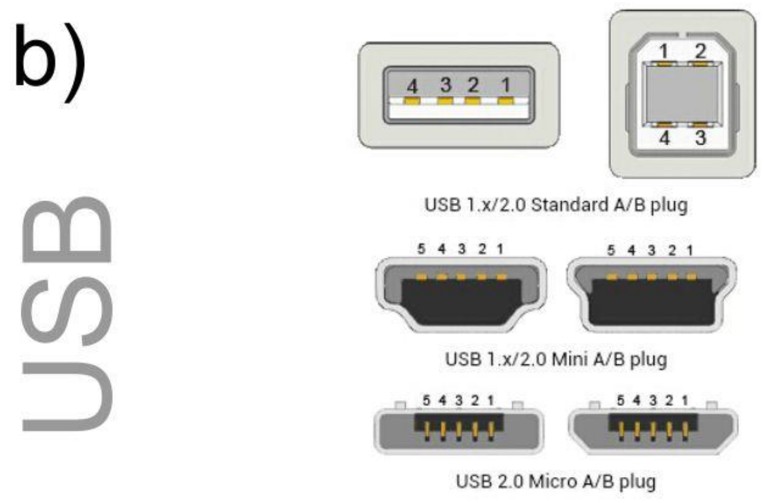
I wanted to compact everything into this case, so I decided to solder some short connectors (HDMI Screen to HDMI Raspberry) and the USB connector of the screen and power supply of the Raspberry. I also put on a switch at the side to disconnect it from the powerbank. It very much depends on how much work you want to invest in this project. There might be also suitable commercial connectors, but I didn't search for them.

Figure 14 shows the typical connectors of the Raspberry, but careful I have 3 or 4 of them and all of them have different connectors (micro USB B, micro HDMI, normal HDMI etc.). Therefore, I just present the assignment of the typical connectors here.
Careful when soldering the HDMI-connectors, because some pins are not connected to the plug (at least in my version). I had to check them (cf. Fig. 14) for continuity manually. Of course you do not have to solder all pins 1:1, I only connected the 7 pins on the upper and 6 on the lower side of the HDMI-connector as shown in Fig. 14.

I connected both USB connectors of the powerbank in parallel to a switch and then to the Raspberry (of course only 5V and GND). This was necessary because one seemed not to be sufficient (although specified to have 2.4 A). The Raspberry then showed a little yellow flash (= power problems) on the screen, which disappeared after connecting both USB ports. To connect the probes conveniently I used a standard D-Sub 9 connector at the side.
I find it still awkward to use the tiny screen keyboard on the small screen with my sausage size fingers. It is ok using a stylus or connecting a keyboard (at least during calibration.
Figure 15 shows the final device (with the cover removed).



| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HDMI TYPE A | 1 | 3 | 4 | 6 | 7 | 9 | 10 | 12 | 13 | 14 | 15 | 16 | 17 |
| HDMI "C" Mini | 2 | 3 | 5 | 6 | 8 | 9 | 11 | 12 | 14 | 17 | 15 | 16 | 13 |



| Pin | | Name | Description | Color |
|---|---|---|---|---|
| Stand. | Mini/Micro | | | |
| 1 | 1 | VCC | +5 V | Red |
| 2 | 2 | D- | Data - | White |
| 3 | 3 | D+ | Data + | Green |
| - | 4 | ID | OTG Identification:<br>• **host:** connected to ground<br>• **slave:** not connected | *Not connected* |
| 4 | 5 | GND | Ground | Black |
| Shell | | Shield | Metal shell | *Drain Wire* |

Fig. 14: Connections of a) HDMI and b) USB connectors.

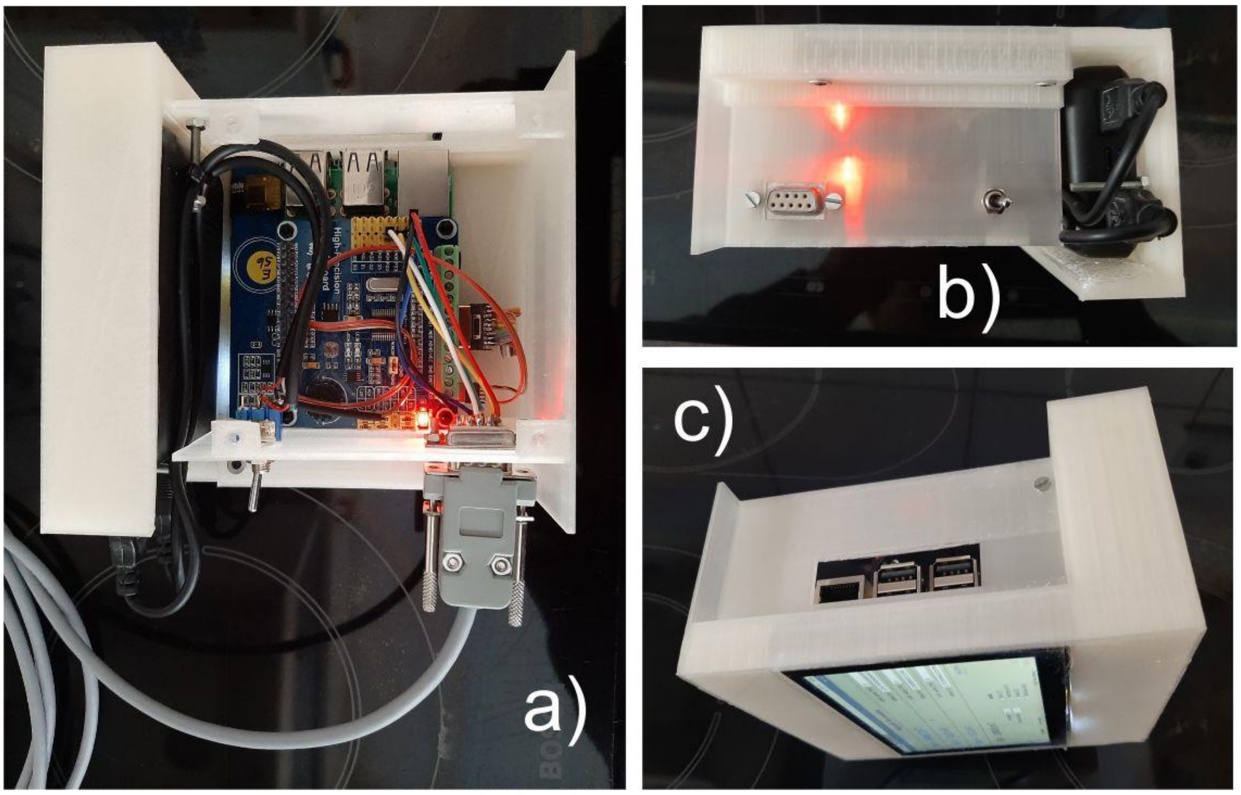


Fig. 15: The final device: a) view from the back (lying on screen). To the left (black box) is the powerbank connected via a regular switch to the USB power port of the Raspberry. To the right you can see my home-made HDMI/HDMI cable. B) Side view showing the D-Sub connector for the Hall-probes. C) shows the other side with the USB ports of the Raspberry.

Packing everything into a mobile, autonomous unit sounds good, but I found the use of the touchscreen awkward. So I also designed just a case for Raspberry and Waveshare-card. So you have to attach a keyboard, mouse and screen. I prefer to use it this way.
You can find the 3D-printer file for this in `3D print\Hallpi simple case.stl`. It looks then like in Fig. 16.

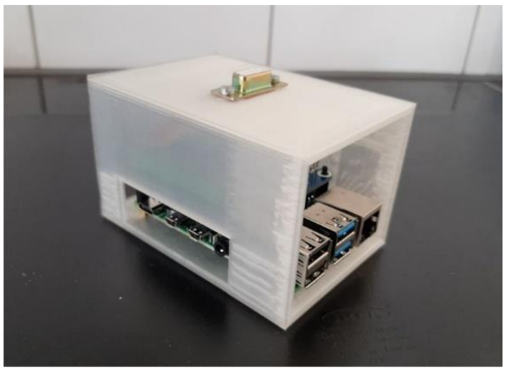


Fig. 16: Simple case for Raspberry and Waveshare-card. One can still attach the probes via a D-Sub connector on the top. Otherwise keyboard, mouse, screen and power supply still need to be attached.

## 8) Calibration

Of course I have calibrated commercial Gaussmeters and tons of permanent magnets, so it is no problem for me to find a suitable arrangement to calibrate my Hall-probe(s).

If you do not have anything to refer to, I suggest to use a Helmholtz-coil. That are two loops of wire with radius *R* and distance *R* to each other. If they have N windings they produce the following flux in the center

$$B(0) = \frac{8\,\mu_0}{5\sqrt{5}}\,\frac{NI}{R} = 8.9918 \cdot 10^{-7} \left[\frac{\mathrm{Tm}}{\mathrm{A}}\right] \frac{NI}{R}$$

if a DC-current of strength I is ran through them. In `3D print\Calibration coil.stl` you find a template for such a coil with R = 22.48 mm giving B = 0.04 mT/A/winding.
Using 1 mm varnished copper wire, I easily managed to get 30 windings on each spool (10 mm wide) and measured a resistance of 0.38 Ω.
Hence, theoretically every amp of current should produce 1.2 mT in its center. However, in reality (due to wire thickness) I measured only 0.98 mT/A.
There are three orthogonal channel in the coil template (see Fig. 17), where you can stick the probe in. They all end right in the center. I managed to put 10 A through it for a short time. Of course you can thinner or thicker wire and a different number of windings. I recommend to calculate the flux in the center then by adding loops using Biot-Savart's law.

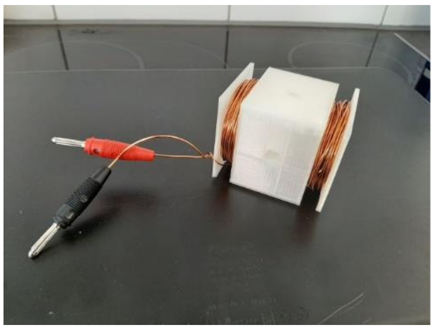


Fig. 17: Calibration coil

Here are the calibration values:

1) 3 axis probe using 3 × CYSJ362A (build in section 1):
x-channel: offset $V_{\mathrm{off}}$ = 1.362 V, calibration constant *A* = -2.26 T/V
y-channel: offset $V_{\mathrm{off}}$ = 1.314 V, calibration constant *A* = 2.96 T/V
z-channel: offset $V_{\mathrm{off}}$ = 1.355 V, calibration constant *A* = -2.76 T/V

2) AST HE244 probe A: offset $V_{\mathrm{off}}$ = 1.721 V, calibration constant *A* = 2.20 T/V

3) AST HE244 probe B: offset $V_{\mathrm{off}}$ = 1.727 V, calibration constant *A* = 1.93 T/V

They show a similar sensitivity like the commercial Hall-sensors I have for a field range of 0.3 to 3T. Only the last digit is a bit unstable… maybe using a grounded shield for all could be an improvement, but something I cannot easily do at home.

## Possible improvements:

1) It would be advantageous to operate the Hall-sensors symmetrically with 5 V. This would improve the sensitivity by a factor 2 because there would no longer be an offset.
   Here is a kit (https://www.henri.de/bauelemente/bausaetze-und-module/strom/28482_/bausatz-netzteil-plus-minus/5v/9v/12v/15v/18v/20v/24v-2x1a-symmetrische-stromv.html) for a symmetric power supply which needs to be combined with a 5V transformer. If you want to run the Raspberry from the +5V output make sure you have at least 2.5 A = 15VAC

2) It would be nice to control the Hallpi-program remotely via the MQTT protocol. This shouldn't be a problem. I just didn't do it.

3) A electromagnetic shielded cover and probe should be tested to see if this further improves the sensitivity/stability.

I hope you enjoyed this little project. If you have any ideas for improvement please let me know.

Stay healthy, yours Peter

## Source Code of Hallpi.py

```
from tkinter import *
from tkinter import filedialog
import ADS1256
import paho.mqtt.client as mqtt
from datetime import datetime
from numpy import array
import socket
import csv

#actual time stamp
now=datetime.now()
time_stamp=now.strftime("%d.%m.%y  %H:%M:%S")
print("Hall-pi  started @ " + time_stamp )
print("Version 1.0 (c) Peter Bluemler, University of Mainz, Germany 4.2020")
print(" ")

#try to get the IP address to establish a MQTT client
hostname = socket.gethostname()
sock=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
try:
    sock.connect(("8.8.8.8",80))
    IPAddr=sock.getsockname()[0]
    print(hostname + " connected via IP-address: "+ IPAddr )
except socket.error:
    print("WARNING: No internet connection ")
    IPAddr=""
if IPAddr !="":  #if there is an IP-address establish MQTT connection
    client = mqtt.Client()
    client.connect(IPAddr,1883,60)
    client.publish("hallpi","Connection established! @ "+time_stamp)

#the global variables used in the program
global B_calib, B_offset, average, digits, unit_mult, comp

#read default values from last leave-procedure from file
filename='defaults.csv'
try:
   with open(filename,'r') as file:
       reader=csv.reader(file)
       data=next(reader)
       B_calib=array([float(data[0]),float(data[1]),float(data[2])])
       B_offset=array([float(data[3]),float(data[4]),float(data[5])])
       average=int(data[6])
       digits=int(data[7])
       unit_mult=float(data[8])
       comp=[int(data[9]),int(data[10]),int(data[11])]
except IOError:
    print("WARNING: could not fine default.csv guessing values")
    B_calib=array([2.0,2.0,2.0])
    B_offset=array([1.1,1.1,1.1])
    average=10
    digits=1
    unit_mult=1
    comp=[1,1,1]

#set the ADC
ADC = ADS1256.ADS1256() # define the ADC
ADC.ADS1256_init()
print("_____________________________________________________")



###############  FUNCTIONS ####################

def read_ADC():
   ADC_volts = array(ADC.ADS1256_GetAll())      #read the voltages from the ADC
   ADC_volts = ADC_volts*5.0/0x7fffff           # multiply all with 5.0 Volts and divide by the bits 2^23
   return ADC_volts


def fetch(entries): # this function handles values entered into the various entry fields
   # 3-5 = calib values, 6-8 offsets, 9=averages, 10=digits
   global B_calib, B_offsets, average, digits
   for k in range(0,len(entries)):
      entry=entries[k]
      if k == 3:
          B_calib[0] =float(entry.get())
          entry.delete(0,END)
          entry.insert(0,B_calib[0])
      if k == 4:
          B_calib[1] =float(entry.get())
          entry.delete(0,END)
          entry.insert(0,B_calib[1])
      if k == 5:
          B_calib[2] =float(entry.get())
          entry.delete(0,END)
          entry.insert(0,B_calib[2])
      if k == 6:
          B_offset[0] =float(entry.get())
          entry.delete(0,END)
          entry.insert(0,B_offset[0])
      if k == 7:
          B_offset[1] =float(entry.get())
          entry.delete(0,END)
          entry.insert(0,B_offset[1])
      if k == 8:
          B_offset[2] =float(entry.get())
          entry.delete(0,END)
          entry.insert(0,B_offset[2])
      if k ==9:
         average=int(entry.get())
         if average < 1:
            average = 1
         entry.delete(0,END)
         entry.insert(0,average)
      if k == 10:
         digits=int(entry.get())
         if digits<0:
            digits=0
         elif digits >8:
            digits=8
         entry.delete(0,END)
         entry.insert(0,digits)


def setzero(entries): #determines the average of 100 measurements at zero field to calc. offsets
    global B_offset
    ave_zero=100      # number of averages
    V_avg=array([0,0,0])
    for k in range(0,ave_zero): #sample flux
        volts=read_ADC()
        V_avg=V_avg+volts[5:8]
    B_offset=V_avg/ave_zero     #calc average
    for k in range(0,len(entries)): #set the entries to the new value
        entry=entries[k]
        if k == 6:
            entry.delete(0,END)
            entry.insert(0,B_offset[0])
        if k == 7:
            entry.delete(0,END)
            entry.insert(0,B_offset[1])
        if k == 8:
            entry.delete(0,END)
            entry.insert(0,B_offset[2])

def set_unit(v):  #set the unit multiplier from the
    global unit_mult
    multiplier={1:1, 2:1000, 3:10000, 4: 10}
    unit_mult=multiplier[v]

def set_comp(c): #just to assign the comp to a global variable
    global comp
    comp=c

def display_flux(entries):
   #read ADC and display calibrated Bxyz and T values
   def check_ADC():
      global B_calib, B_offset, average, digits, unit_mult,comp
      V_avg=array([0,0,0])
      for j in range(0,average):    #do the averaging
           volts=read_ADC()
           V_avg = V_avg + volts[5:8]
      B_act=V_avg/average
      Bstr=["Bx","By","Bz"]   #define output string
      Bamp=0
      for k in range(0,3):
          entry=entries[k]
          if comp[k]==1:  # if checkbutton is ON
              pubstr=Bstr[k] + " = "+"% "+str(digits+7)+"."+str(digits)+"f"  #compose output string
              pubstr=pubstr% round((B_act[k]-B_offset[k])*B_calib[k]*unit_mult,digits)
              Bamp=Bamp+((B_act[k]-B_offset[k])*B_calib[k]*unit_mult)**2
              if unit_mult==1:
                  pubstr=pubstr + " T"
              if unit_mult==1000:
                  pubstr=pubstr + " mT"
              if unit_mult==10000:
                  pubstr=pubstr + " G"
              if unit_mult==10:
                  pubstr=pubstr + " kG"
              entry.config(text=pubstr)
              if IPAddr !="":      # if connected via MQTT write string to client
                 client.publish("hallpi",pubstr)
          else:
              pubstr=Bstr[k] + " = ---- " #if checkbutton is not ticked display ----
              entry.config(text=pubstr)
      #calc and display B amplitude
      entry=entries[11]
      Bamp=Bamp**0.5
      pubstr="|B| = "+"% "+str(digits+7)+"."+str(digits)+"f"
      pubstr=pubstr% round(Bamp,digits)
      if unit_mult==1:
         pubstr=pubstr + " T"
      if unit_mult==1000:
         pubstr=pubstr + " mT"
      if unit_mult==10000:
         pubstr=pubstr + " G"
      if unit_mult==10:
         pubstr=pubstr + " kG"
      entry.config(text=pubstr)
      entry.after(1,check_ADC)
   check_ADC()

def load_probe(window):  #read in the data writen with save_probe
    global B_calib, B_offset, average, digits, unit_mult, comp
    filename=filedialog.askopenfilename(title="Save settings as", filetypes=(("csv files","*.csv"),("all files","*.*")))
    try:
       with open(filename,'r') as file:
           reader=csv.reader(file)
           data=next(reader)
           B_calib=array([float(data[0]),float(data[1]),float(data[2])])
           B_offset=array([float(data[3]),float(data[4]),float(data[5])])
           average=int(data[6])
           digits=int(data[7])
           unit_mult=float(data[8])
           comp=[int(data[9]),int(data[10]),int(data[11])]
    except IOError:
        print("ERROR reading "+filename+"   values not changed!")


def save_probe(window):
    global B_calib, B_offset, average, digits, unit_mult, comp
    filename=filedialog.asksaveasfilename(title="Save settings as", filetypes=(("csv files","*.csv"),("all files","*.*")))
    file=open(filename,'w')  #save settings to default
    with file:
       writer=csv.writer(file)
       data=list(B_calib)+list(B_offset)+[average,digits,unit_mult]+comp
       print("saved the following data to "+filename)
       print(" ")
       writer.writerow(data)


def leave(entries):
   global B_calib, B_offset, average, digits, unit_mult, comp
   file=open('defaults.csv','w')  #save settings to default
   with file:
       writer=csv.writer(file)
       data=list(B_calib)+list(B_offset)+[average,digits,unit_mult]+comp
       print("saved the following data to default.csv")
       print(" ")
       writer.writerow(data)
   quit()


def makeform(window):
   #MAIN DISPLAY OF FIELD
   global B_calib, B_offset, average, digits, unit_mult, comp
   B_act=read_ADC()
   entries = []
   # draw some boxes
   Mc1=Canvas(window,width=800,height=440)
   Mc1.place(x=1,y=1)
   Mc1.create_rectangle(5,40,795,110)
   Mc1.create_rectangle(5,40,795,180)
   Mc1.create_rectangle(5,40,795,250)
   lab_dummy=Label(window,text='Magnetic flux monitor', fg='magenta4', font=("Arial 16 bold")).place(x=290,y=8)
   lab_dummy=Label(window,text='© 2020 Peter Blümler', font=("Arial 8")).place(x=600,y=420)
   #  the flux labels Bx,y,z
```

```
lab_Bx=Label(window,text='Bx = '+str(round((B_act[5]-B_offset[0])*B_calib[0],digits))+' T', fg='blue', font=("Arial",24),justify=LEFT)
lab_Bx.place(x=50,y=55)
lab_By=Label(window,text='By = '+str(round((B_act[6]-B_offset[1])*B_calib[1],digits))+' T', fg='blue', font=("Arial",24),justify=LEFT)
lab_By.place(x=50,y=125)
lab_Bz=Label(window,text='Bz = '+str(round((B_act[7]-B_offset[2])*B_calib[2],digits))+' T', fg='blue', font=("Arial",24),justify=LEFT)
lab_Bz.place(x=50,y=195)
lab_Ba=Label(window,text='|B| = '+str(round((B_act[7]-B_offset[2])*B_calib[2],digits))+' T',  font=("Arial",24),justify=LEFT)
lab_Ba.place(x=50,y=265)
#calibration values of ADC -> magnetic field
lab_Bcx=Label(window, width=16, text='calib. value [T/V]:', justify=RIGHT).place(x=565,y=51)
ent_Bcx=Entry(window, width=12)
ent_Bcx.place(x=690,y=50)
ent_Bcx.insert(0, B_calib[0])
lab_Bcy=Label(window, width=16, text='calib. value [T/V]:', justify=RIGHT).place(x=565,y=121)
ent_Bcy=Entry(window, width=12)
ent_Bcy.place(x=690,y=120)
ent_Bcy.insert(0, B_calib[1])
lab_Bcz=Label(window, width=16, text='calib. value [T/V]:', justify=RIGHT).place(x=565,y=191)
ent_Bcz=Entry(window, width=12)
ent_Bcz.place(x=690,y=190)
ent_Bcz.insert(0, B_calib[2])
#offset values of ADC -> magnetic field
lab_Box=Label(window, width=16, text='offset [V]:', justify=RIGHT).place(x=540,y=81)
ent_Box=Entry(window, width=18)
ent_Box.place(x=642,y=80)
ent_Box.insert(0, B_offset[0])
lab_Boy=Label(window, width=16, text='offset [V]:', justify=RIGHT).place(x=540,y=151)
ent_Boy=Entry(window, width=18)
ent_Boy.place(x=642,y=150)
ent_Boy.insert(0, B_offset[1])
lab_Boz=Label(window, width=16, text='offset [V]:', justify=RIGHT).place(x=540,y=221)
ent_Boz=Entry(window, width=18)
ent_Boz.place(x=642,y=220)
ent_Boz.insert(0, B_offset[2])
#average entry
lab_avg=Label(window, width=10, text='averages:', justify=RIGHT)
lab_avg.place(x=300,y=330)
ent_avg=Entry(window, width=6)
ent_avg.place(x=380,y=330)
ent_avg.insert(0, average)
#decimal places
lab_dig=Label(window, width=14, text='decimal places:', justify=RIGHT)
lab_dig.place(x=265,y=355)
ent_dig=Entry(window, width=6)
ent_dig.place(x=380,y=355)
ent_dig.insert(0, digits)
#set zero button
zero_but=Button(window, text='set Zero', fg='blue', command=lambda: setzero(entries))
zero_but.place(x=710,y=260)
#set load_probe button
```

```
   load_but=Button(window, text='load probe',  command=lambda: load_probe(window))
   load_but.place(x=100,y=405)
   #set save_zero button
   save_but=Button(window, text='save probe',  command=lambda: save_probe(window))
   save_but.place(x=230,y=405)
   #quit button
   quit_but=Button(window, text='QUIT', fg='red', command=lambda: leave(entries))
   quit_but.place(x=732,y=405)
   #
   #Finally the interacting elements of the GUI are defined in an entry-list
   entries.append(lab_Bx)     # [0] = Bx main display label
   entries.append(lab_By)     # [1] = By main display label
   entries.append(lab_Bz)     # [2] = Bz main display label
   entries.append(ent_Bcx)    # [3] = Entry field of Bx calibration
   entries.append(ent_Bcy)    # [4] = Entry field of By calibration
   entries.append(ent_Bcz)    # [5] = Entry field of Bz calibration
   entries.append(ent_Box)    # [6] = Entry field of Bx offset
   entries.append(ent_Boy)    # [7] = Entry field of By offset
   entries.append(ent_Boz)    # [8] = Entry field of Bz offset
   entries.append(ent_avg)    # [9] = average label
   entries.append(ent_dig)    # [10] = digits label
   entries.append(lab_Ba)     # [11] = amplitude of field
   return entries


# The main loop
if __name__ == '__main__':
   window = Tk()
   v=IntVar()
   # define unit multiplactor for radiobuttons
   if unit_mult==1:
      v.set(1)
   if unit_mult==1000:
      v.set(2)
   if unit_mult==10000:
      v.set(3)
   if unit_mult==10:
      v.set(4)
   units=[("Tesla [T]",1),("milliTesla [mT]",2),("Gauss [G]",3),("kiloGauss [kG]",4)]
   #make window
   window.title('Hall-pi flux monitor')
   window.geometry("800x440+0+0") #adapted to the 5" screen (800 * 440 pixels) 0+0 is top left corner
   ents = makeform(window)        #make the GUI layout and get the entries
   # make the radiobutton field for the units
   rad_lab=Label(window,text="units:", font=('Arial 12 bold')).place(x=525,y=305)
   for txt, val in units:
       rad_but=Radiobutton(window,text=txt, variable=v, command=lambda: set_unit(v.get()),value=val)
       rad_but.place(x=500,y=310+val*20)
   # and checkbuttons for the flux components (could not find a more elegant way)
   csx=IntVar()
```

```
csx.set(comp[0])
csy=IntVar()
csy.set(comp[1])
csz=IntVar()
csz.set(comp[2])
chBBx=Checkbutton(window, variable=csx, command=lambda: set_comp([csx.get(),csy.get(),csz.get()])).place(x=15,y=63)
chBBy=Checkbutton(window, variable=csy, command=lambda: set_comp([csx.get(),csy.get(),csz.get()])).place(x=15,y=133)
chBBz=Checkbutton(window, variable=csz, command=lambda: set_comp([csx.get(),csy.get(),csz.get()])).place(x=15,y=203)
comp=[csx.get(),csy.get(),csz.get()]
#
# in case return is pressed after changing entries they are passed to function fetch
window.bind('<Return>', (lambda event, e=ents: fetch(e)))
display_flux(ents)  # measure and display flux
window.mainloop()   # loop
```