Updated: January 18, 2022

User Guide for the Laboratory

TTT4280 - Sensors and Instrumentation

Spring 2022

Welcome to the Laboratory for TTT4280: Sensors and instrumentation. This guide will cover everything you need to know. If you have any questions, you can contact your stud.ass or vit.ass.

NTNU
Faculty of Information Technology
and Electrical Engineering
Department of Electronic Systems

Contents

Chapter 1

Setup of the Raspberry Pi

1.1 Introduction

The Raspberry Pi will serve as the basis of the instrumentation system and provide the interface between hardware and software. First we must install the Linux-based operating system and configure the network connection.

Both Linux in general and the Raspberry Pi have a large user base. You will find much documentation and countless of tutorials available online. If you are stuck or in doubt, a search will definitely provide answers. We will base this guide on the well written, official Raspberry Pi documentation to not re-invent the wheel. Links will be provided to all steps.

To make the experience as smooth as possible, we will refrain from dragging monitors and keyboards into the lab. Instead, you will configure a network connection directly on the SD card and connect to the Raspberry Pi via SSH. This is called a *headless setup*.

1.2 Installing the OS

This section will cover the OS installation process.

The easiest way of installing Raspberry Pi OS is to use the Raspberry Pi Imager, available here. It lets you set up `ssh`, described in Section 1.4, if you press Ctrl -⇧ Shift -X , among other things.

1.2.1 Downloading the OS

Download a copy of Raspberry Pi OS from the official download page. Three options are available: With desktop and additional software, only desktop or the minimal installation. Go for the first one if in doubt. If you know what you are doing, you can go for the minimal installation.

1.2.2 Writing the ISO image to the SD card

Unzip the ISO image you downloaded. This is a file containing a disk image, which you can think of as clone of a physical drive or disc. The ISO file has to be written to the SD card using dedicated software.

Follow the instructions in the official documentation on how to write the image to the SD card, using either balenaEtcher or one of the other options. The former is preferred.

*Be **extremely** careful when selecting the target device for flashing. Your operating system may just let you overwrite your entire hard drive, **even if the OS is running from said drive.***

Do not boot the device before doing the steps in the next section.

1.3 Network connection

Without any peripherals such as a screen or keyboard, we need a working network connection to be able to communicate with our Raspberry.

We have three alternatives to use for connecting to the Raspberry. You either connect the Raspberry to your laptop directly with Ethernet, or you use your phone as a hotspot. If you are at home, you can also connect the Raspberry to your router via Ethernet or use WiFi. Of course, there is no need to stick to either of these. Read through all options and feel free to test them both. For the lab sessions, we suggest that you use your phone as a hotspot.

We strongly discourage using Eduroam via Ethernet of WiFi for security reasons.

1.3.1 Wireless connection (recommended)

The preferred alternative is to use the hotspot on your phone to create the network. This eliminates the cable and is a bit more flexible, as well as allowing for more users to connect to the Pi. We will follow the instructions form https://www.raspberrypi.org/documentation/configuration/wireless/headless.md.

We won't need to download any software to the Raspberry Pi, but the surfing you do with your laptop while connected to the hotspot will impact your data plan. Note that data sent between your laptop and the Raspberry will not affect your data plan.

To make the Raspberry connect to your phone's hotspot, you first have to change some settings. This can be done directly on the SD card.

Navigate to the mounted SD card and create the following file.

```
/boot/wpa_supplicant.conf
```

The *WPA supplicant* is a service that handles WiFi authentication. Open the file in a text editor, such as Vim, Emacs, Nano, Atom, VS Code, Notepad or Notepad++. Do not use a word processor like Microsoft Word – the file must be plain text.

Fill in your network credentials as follows:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=NO

network={
ssid="network-name"
psk="network-password"
}
```

You can add as many networks as you want by creating new *network* sections encased in brackets. Feel free to add your home WiFi settings if you want to work with the projects outside the lab.

Your phone will list all connected devices with hostnames and IP addresses under the network settings. The Pi should show up here with the default hostname **raspberrypi**.

Be careful if you copy/paste code from a PDF or website. The formatting may prevent the code from working.

1.3.2 Ethernet connection

You can also connect the Raspberry Pi directly to your laptop with Ethernet. This requires you to enable the connection on your laptop by hanging the networking settings, which is a different process depending if you are using Windows, Mac or Linux. You may also set up a static IP address on the Ethernet port of your laptop. Both Ethernet cables and Ethernet-to-USB adapters are provided in the lab. Simply connect the Ethernet cable to the Raspberry and your laptop.

If your computer supports it, you can also share your internet connection to the Raspberry Pi.

Note that connecting the Raspberry to your laptop via Ethernet disables your internet access when on Eduroam, for security reasons.

1.3.3 Connecting to a router

If you are at home, you can simply connect your Raspberry Pi to your router via Ethernet. Alternatively, you can connect to the router with WiFi as described above.

1.4 SSH

SSH is a protocol which lets you log on to a remote machine directly in your terminal.

We will first try to connect to the Raspberry with the hostname. If that doesn't work, we need to find the IP address first.

1.4.1 Enable SSH on the Raspberry

As mentioned before, a Raspberry Pi with default credentials is a large security risk, as anybody who knows them could gain access to it. This is why Raspberry disables SSH by default. Turn it on by placing an empty file called `ssh` inside the `/boot/` folder on the SD card. It is important that the file is empty and has no file extension.[1]

1.4.2 Connecting to the Pi

Power up the Pi, enable your hotspot or plug in your Ethernet cable. For **Linux and Mac**, SSH works out of the box. **Windows** users can use PowerShell to connect to the Pi. Open a terminal (or PowerShell window) and type

```
ssh pi@raspberrypi.local
```

where *raspberrypi* is the default hostname of the Raspberry and *pi* is the default user. The *.local* works much like normal toplevel domain, such as *.com* or *.no*, but tells your computer to have a look in the local network. If successful, you will be prompted for a password. The default password is *raspberry*. Note that Linux systems hide passwords to prevent others from determining their length. Confirm with enter.

If no device with the matching hostname could be found (hostname could not be resolved), you need to first find and connect with the IP-address. This is usually caused by mDNS being disabled on the network, broadcasting messages being blocked on the network, or your machine lacking a mDNS client. After finding the IP-address, simply connect using

```
ssh pi@ip-address
```

After the first connection, you should be able to use the hostname.

For more info, check the official documentation.

1.4.3 Finding the IP address

Finding the IP address depends on which connection method you used. However, we recommend that you use nmap as it should work in most cases.

[1] Windows Explorer might hide known file extensions. To display them, open the View tab in Explorer and check the box "File Extensions" in the "Show/Hide" pane. This is known to work on Windows 10. Command-line utilities, like `cmd` or PowerShell, do show file extensions.

Scanning the network with nmap (recommended)

nmap is a versatile and handy tool that simply checks a range of IP adresses and reports which are being used. It works for all platforms. The steps are taken from the Raspberry Pi documentation.

The nmap command (Network Mapper) is a free and open-source tool for network discovery, available for Linux, macOS, and Windows.

To install on Linux, install the nmap package e.g. apt install nmap.

To install on macOS or Windows, see the nmap.org download page. To use nmap to scan the devices on your network, you need to know the subnet you are connected to. First find your own IP address, in other words the one of the computer you're using to find your Pi's IP address:

- On Linux, run `hostname -I` in a terminal window
- On macOS, go to System Preferences then Network and select your active network connection to view the IP address
- On Windows, go to the Control Panel, then under Network and Sharing Center, click View network connections, select your active network connection and click View status of this connection to view the IP address; or run `ipconfig` in PowerShell or `cmd`.

Now that you have the IP address of your computer, scan the whole subnet for other devices. For example, if your IP address is 192.168.1.5, other devices can be at addresses like 192.168.1.2, 192.168.1.3, 192.168.1.4, etc. The notation of this subnet range is 192.168.1.0/24 (this covers 192.168.1.0 to 192.168.1.255).

Now use the nmap command with the -sn flag (ping scan) on the whole subnet range. This may take a few seconds:

```
nmap -sn 192.168.1.0/24
```

Checking the hotspot settings for an IP

Some flavors of Android provide a list of clients along with their IP addresses in the Wi-Fi hotpot settings. This is a handy shortcut if you can use it. Others will simply provide a count of the connected devices and nothing more.

iOS will not provide the user with a list of devices.

Fing app

The Fing app is a free network scanner for smartphones, available for Android and iOS. It will generate a list of devices on a network. If you are using a hotspot, then you need to install the app on another phone than the one that runs the hotspot.

Ethernet to host computer

Using a direct Ethernet connection, the IP of the Raspberry should be displayed somewhere in your network settings, depending on your OS. For Mac and Linux users, the kernel log should have an entry. Open a terminal and write **dmesg** and read through the last entries.

Router

If you have access to the router, you can easily find a list of clients on the router's maintenance page. You can find that by navigating to the routers gateway address with a web browser, typically `192.168.0.1`. The gateway address is usually written on a sticker. If not, there are various ways to find it, depending on your OS.

1.5 Change the hostname

A hostname acts the same way as domain names, like `ntnu.no`. It is a name given to a device that you can use in place of the IP-address when you want to connect to it. This can easily be changed by editing the file

```
/etc/hostname
```

The default hostname is *raspberrypi*. Hostnames should be lowercase, alphanumerical and single worded. Cool hostnames include, but are not limited to, *hal9000*, *apollo*, *kermit*, *einstein* or *uran238*.

1.5.1 Change the password

The first thing we do is change the password for the user pi from the default password. An unsecured Raspberry Pi with default credentials connected to a network represents a major security risk. To change the password of the current user, type the command

```
passwd
```

As mentioned before, Linux hides passwords completely when you write them. This prevents outsiders from counting the number of characters in the password.

1.6 File transfer using SFTP

SFTP (SSH File Transfer Protocol) uses the SSH connection to transfer files between devices. As SSH is end-to-end encrypted, it is very secure but impractical for transferring large files. We will use SFTP later to transfer the ADC data from the Pi to you machine for further processing. You can use `scp` from your computer (Windows users, use PowerShell if you do) to do this, but SFTP is more intuitive.

To use SFTP, we first need a client. Download them and have a glance at the manual when in doubt.

- For Windows, use WinSCP or Cyberduck.
- For MacOS, use Cyberduck.
- For Linux, download Filezilla from the software repository with `apt install filezilla`.

We discourage the use of FileZilla on Windows and iOS, because the installer is bundled with adware.

The user interfaces for the clients differ, but the procedure is more or less the same. Fill in the hostname or IP of the raspberry, as well as login credentials. The port for SSH is 22 by default. Connect and start dragging files between your laptop and the Raspberry.

Note that your user's home directory is `/home/pi/`. This is *your* folder and you are free to create folders and files here.

1.7 Create a simple Python script

Congratulations, you are in! Now, create a simple Python script and you will be ready to start the lab.

First have a look what content is in your current folder by typing

```
ls
```

Now we want to make a new directory with

```
mkdir mydir
```

Enter your new directory with

```
cd mydir
```

and exit with

```
cd ..
```

Let's create your first script with

```
nano test.py
```

This will open a file `test.py` with the *nano* text editor. If the file doesn't exist in the current directory, it will be created.

Now create your simple program.

```
print("Hello World!")
```

Exit nano by hitting **Ctrl+X** and proceed with a **Y** to save the file. Run the program by typing

```
python3 test.py
```

If you want a quick peek inside a file, you can print the contents by using the easy-to-remember

```
cat test.py
```

Note that you should always specify that you want to use Python 3. Python 2 is nearing its end-of-life, but it's still sticking around. So avoid the confusion and be consistent.

1.8 Optional: Connecting to the graphical interface using VNC

If connecting the a server with SSH and using terminals still is not your cup of tea, you can easily access the graphical interface by using VNC. VNC is a protocol that lets you connect to another computer's graphical interface, like SSH lets you connect to its terminal. This is neat if you want to work on something using graphics, but cannot be bothered to set up a monitor, which you shouldn't either way.

There exist many VNC clients. We recommend using the RealVNC Chrome Extension. Install it and provide it with the hostname of the Raspberry.

The Raspberry should already be running a VNC server, if you have installed one of the versions with a desktop interface.

1.9 Further reading

Here are some more links which may come in handy later on:

- Linux Command Line Cheat Sheet

Chapter 2

Connecting the ADC

The instrumentation setup we will use is based on five analog to digital converters (ADC) of type MCP3201. The MCP3201 is a single-channel ADC with 12 bits resolution, and we will run them in parallel for sampling five channels simultaneously. In addition, we need other circuitry like filters and amplifiers for noise reduction, signal filtering and/or amplification.

The datasheets for all active components used in our lab setup can be found on Blackboard. Use them to learn how they work and how they should be wired up.

Also use this and the other lab manuals actively. They are full of tips and tricks that will make your life easier.

2.1 Raspberry Pi GPIO Pins

The Raspberry Pi 3B has a 40-pin connector where all the GPIO pins are located. There are 26 GPIO pins in total, but they are not numbered according to the header pin numbering. Figure 2.1 shows an overview of the 40-pin header and what each of the physical pins are connected to. This can be compared with the naming that you will find on the Pi Wedge breakout board.

2.2 The Pi Wedge Breakout Board

The Pi Wedge breakout board has an entire hookup guide on Sparkfun, that you can find here if you want more information. Omega Verksted provides an alternative breakout board by Makerfabs. It looks a bit different and is slightly larger, but apart from that it works like the one used in the following example.

Make sure to connect the ribbon cable the correct way. The cable should point away from both the Wedge and the RPi. On the Wedge, there is a notch in the connector, which makes it impossible to plug in the cable the wrong way (unless you force it hard). The corresponding notch in the other end should then point inwards on the RPi board. Figure 2.2 summarizes how the connection should look like.

The Pi Wedge has all ground pins of the header connected together on a ground plane, and the 5 V and 3.3 V pins are connected together with decoupling capacitors to ground. It is recommended to add more decoupling capacitors in parallel to improve stability and reduce noise on the supply lines. The schematic drawings for the RPi and Pi Wedge are available on Blackboard.

In our experience, it is quite hard to remove the connector from the RPi, and much easier to remove it from the Pi Wedge. So when detaching the RPi for transportation, just wrap the ribbon cable carefully around it.

All the instrumentation circuitry will be run on 3.3 V potential to avoid complications with the logic levels of the RPi GPIO pins. However, the radar module needs 5 V, so the active bandpass filter will therefore also be used to change the DC voltage offset from the radar signals down from 2.5 V to 1.67 V before sampling the data with the ADCs.

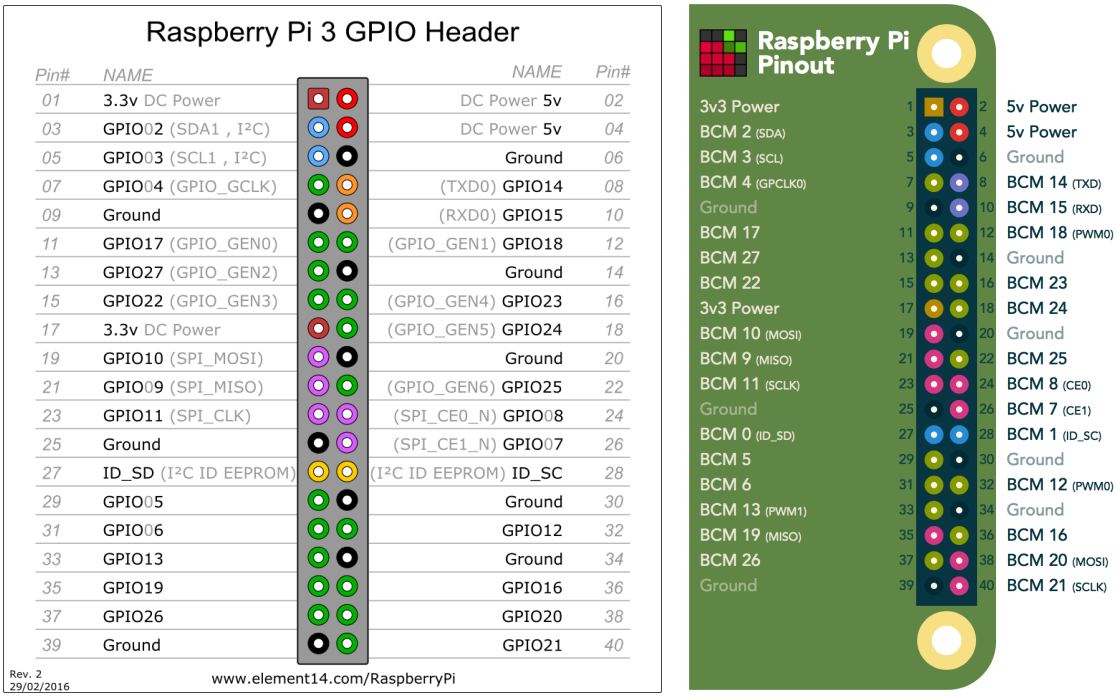


Figure 2.1: Two variants of the Raspberry Pi 3B 40-pin header. GPIO or BCM (chip pin names) and hardware module names are listed alongside the respective physical pins. Figure courtesy of Farnell/Element14 (left) and Raspberry Pi Pinout. (right). Check out the latter one, lots of graphical info!

2.2.1 General Advice

The manuals should provide most of the information you need in order to get the overall system wired up and running in a good way. You are of course free to choose other ways if you like to test and figure things out yourselves. In any case, you should always consult the component datasheets to find specific information and recommended layout considerations for example.

We expect you to use electrical components, wires and breadboards from previous courses in the physical implementation of the circuits. If you find that you need something that is not listed in the lab kit, use what you have or visit Omega Verksted to buy more components on your own initiative. We are very fond of original solutions and people who want to do things in interesting ways!

A general tip is to try to keep high frequency wiring as short and well organized as possible. They will work as antennas for noise in and out of your circuits. You should also take care in the way you connect ground and supply points and avoid creating ground or supply loops. Such loops will soak up noise and can be a source of great frustration when strange things start to happen for no apparent reason.

Finally, try your best to separate digital and analog parts of the circuit. This might not be easy on a prototype breadboard, but should be done to the best of your ability. Digital signal lines carry lots of dirty signals into our circuit, and should be kept away from analog circuitry as best one can.

2.3 ADC Control

One of the key issues when using an ADC to convert some electrical signal from a sensor is to know when and how often a sample or output signal is taken/sent. This is not so easy when we are running our code on a computer with a high level operating system. The Linux kernel employs a scheduler that allocates a certain time slot in sequence for all processes that are running on the system. Hence, our code might be halted at any time, and resumed several milliseconds later if there is much else to do. This is of course not good if we want to sample a sensor live and indefinitely with a high sampling frequency.

We are going to solve this using Direct Memory Access (DMA) which will make the hardware

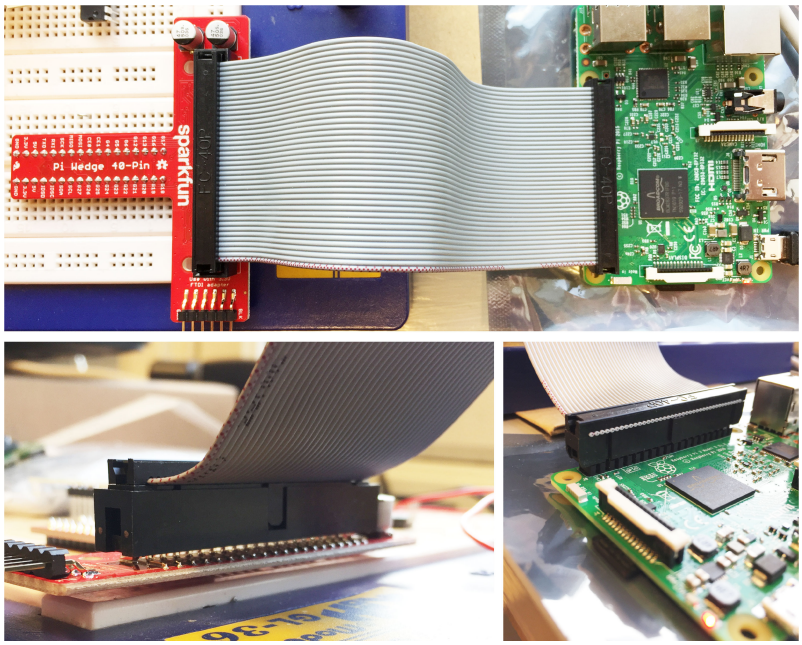


Figure 2.2: How to connect the Pi Wedge breakout board.

modules, located in the periphery of the actual CPU, sample and control the pins without interference from the CPU itself. The data from each sample will continuously be stored directly into the memory in a ring buffer, and our control program running on the CPU will only run for a short time every millisecond to pull the data out from the buffer before new samples overwrite them. Hence, we only need CPU time in small slots every now and then and will not be interrupted without knowing it.

2.3.1 Low Level Hardware Register Control

We will use a C library called *pigpio* (read as pi-gpio) to communicate with the ADCs using DMA. This is available from http://abyz.me.uk/rpi/pigpio/.

If your Pi has an Internet connection, download pigpio with the command below. If not, download it to your computer and transfer it to the Pi with e.g. SFTP. Extract the archive when it is downloaded to the Pi.

```
wget https://github.com/joan2937/pigpio/archive/master.zip
unzip pigpio.zip
```

Once the zip archive is extracted, move to the folder it was extracted to, then compile and install pigpio:

```
cd pigpio
make
sudo make install
```

The command `make` will use the file `Makefile` to figure out how it should automatically run `gcc` for you in order to produce a working library. `make install` will then use the same `Makefile` to figure out what are the main parts of the library (some header files, a library file) and install them to `/usr/local/lib` and `/usr/local/include`.

You are now in principle ready for sampling on the RPi side. We thus move on to the circuitry.

2.4 Instrumentation Circuit

The instrumentation is simple, using five ADCs in parallel. They should use a common wire/signal for the clock and chip select signals, and five separate wires for the data return paths (MISO signals).

A working example of the instrumentation circuit will be available in the lab room for inspiration.

Figure 2.3 shows an example of the **general layout** of the system, not including decoupling capacitors. You have to chose your own inputs for signals to the raspberry pi, the Wires labelled "DATA(1-5)".

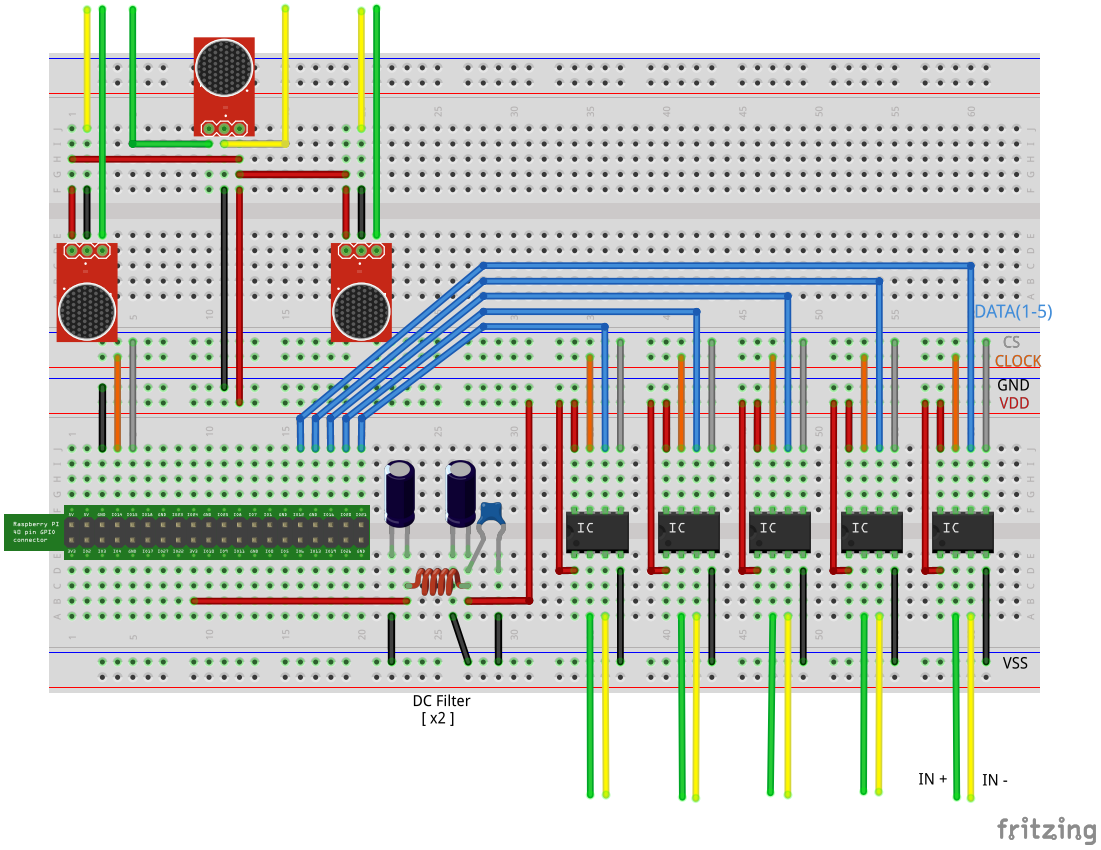


Figure 2.3: The system as a whole should look something like this, picture does not include decopling capacitors.

2.4.1 Connecting the Supply Line Noise Reduction Filter

Noise on the supply lines in an electronic system is a common source of failures and frustration when it comes to finding out what is wrong in a circuit. Especially the power supply to analog circuits (op-amps, microphones etc) needs to have as little noise and ripple as possible. See for example the app-note on Bypassing and Decoupling uploaded to Blackboard.

To separate our digital and analog parts of the circuit to the best of our ability, we will use a Pi bridge, or DC filter on the 3.3 V supply line for the analog circuits, illustrated in Figure 2.4. This filter consists of large stabilizing capacitors, a large inductor for high frequency suppression, and a small capacitor for high frequency shunting.

Wire up this filter close to the analog 3.3 V supply and ground pins on each side of the Pi Wedge. Place the capacitors on the right side in the supply rails along the sides of the breadboard.

This should ensure that we at least stabilize the analog supply network properly and reduce the

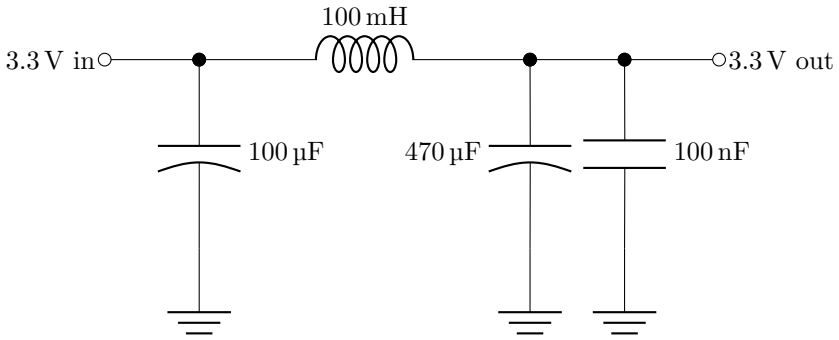


Figure 2.4: Schematic sketch of a supply line noise reduction block.

noise to an acceptable level. See Figure 2.5 for a physical implementation.

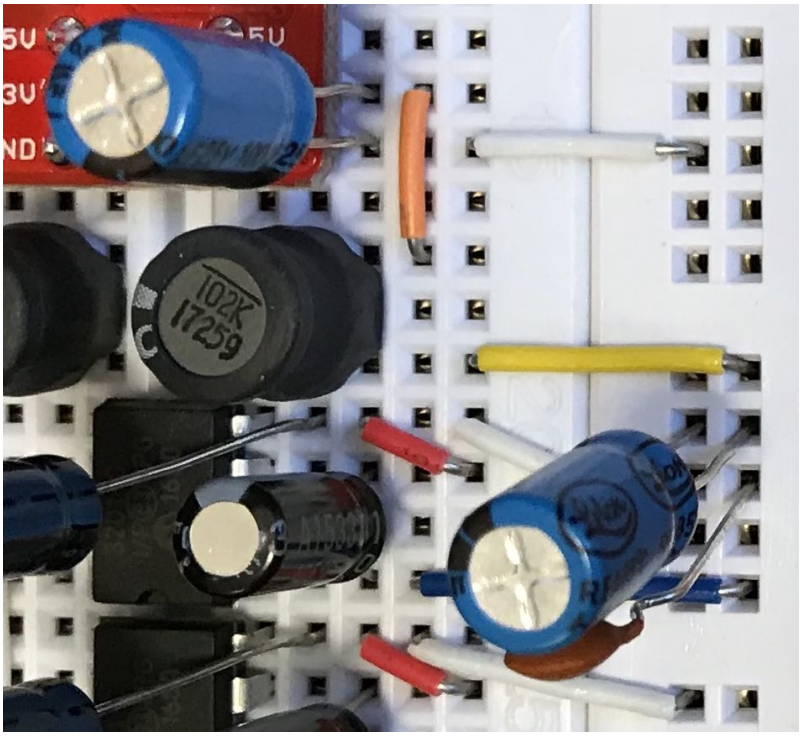


Figure 2.5: Physical implementation of supply line noise reduction filter.

2.4.2 Connect the ADCs

Make sure you look into the datasheet to check the physical pin layout for the ADCs, and do not wire things live (i.e with power connected) unless you know well what you are doing. It is also recommended to start with one ADC and verify that it works before you move on and add more complexity to your system. This makes the system easier to debug.

Mount the five ADCs next to each other. Choose one side of the board for the digital lines. Then start to wire together the chip select and clock signal pins together with short and tight fitting wires. Connect the ADCs iteratively, so leave all analogue inputs open at first.

When the common digital lines are connected together, continue with connecting the supply pins, V_{dd} to 3.3 V from the digital supply side, and the ground pins V_{ss} to GND from the analog supply side. Due to the layout on a breadboard, it is not so easy to separate the ADCs to a digital GND and an analog GND, so just use a common GND on the analog side. Have a look at Figure 2.7 for inspiration. Wire the GND signals from all V_{ss} pins to the negative analog input pins, $IN\_$, separately for all ADCs. Also, add distinct 3.3 V signals from the analog side to each ADC's V_{ref}

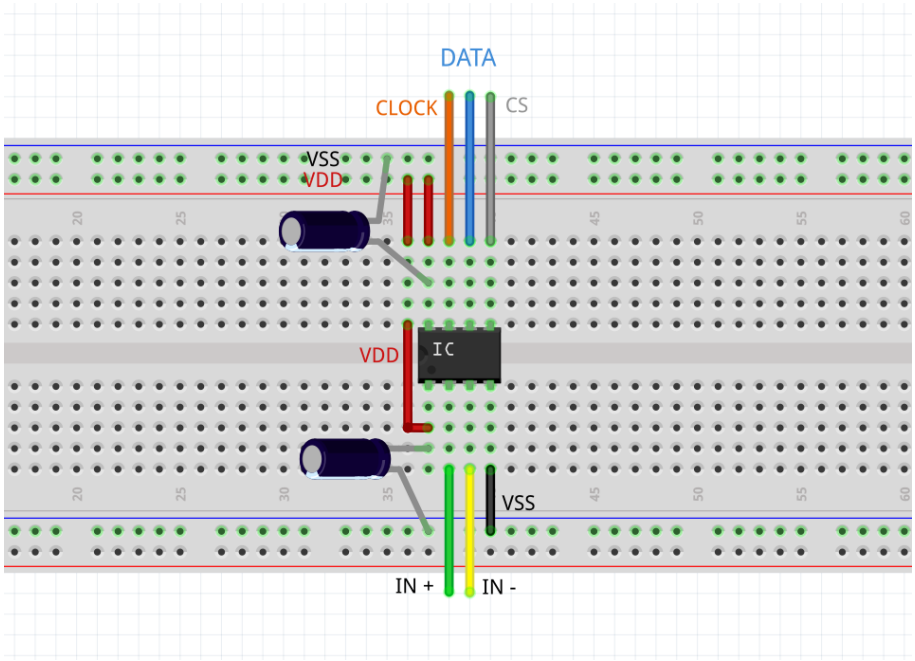


Figure 2.6: Model of the connection setup. Each ADC needs its own decoupling of the power line.

pin.

Add the 1 µF capacitors directly between all V_{dd} and V_{ss} pins, and the 10 µF capacitors between the V_{ref} and IN_- pins of each ADC.

Then connect the digital lines to the correct digital inputs/outputs and to a corresponding GPIO pin on the Pi Wedge.

The instrumentation system should be ready for testing, but *double check that you have connected things correctly*!

The final result should look something like Figure 2.8.

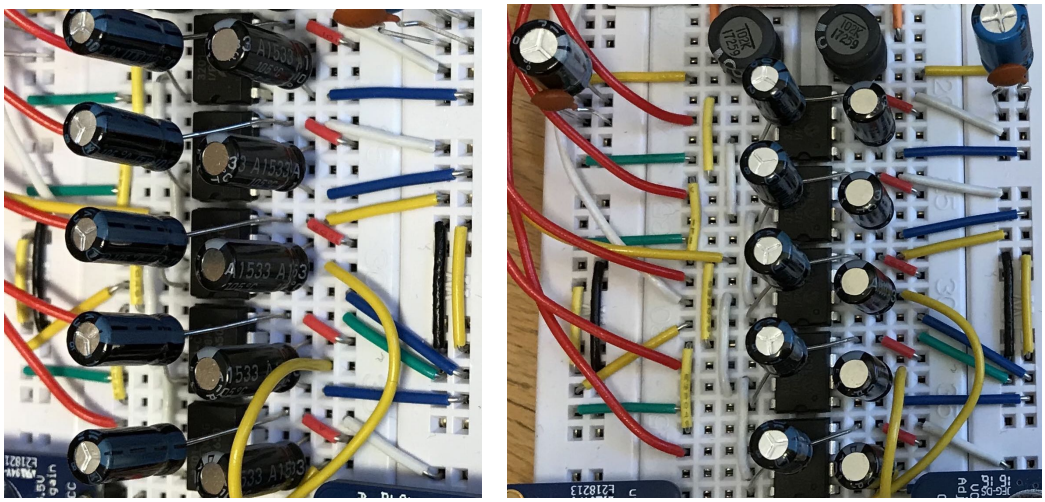


Figure 2.7: ADCs wired up with all signals and stabilizing capacitors.

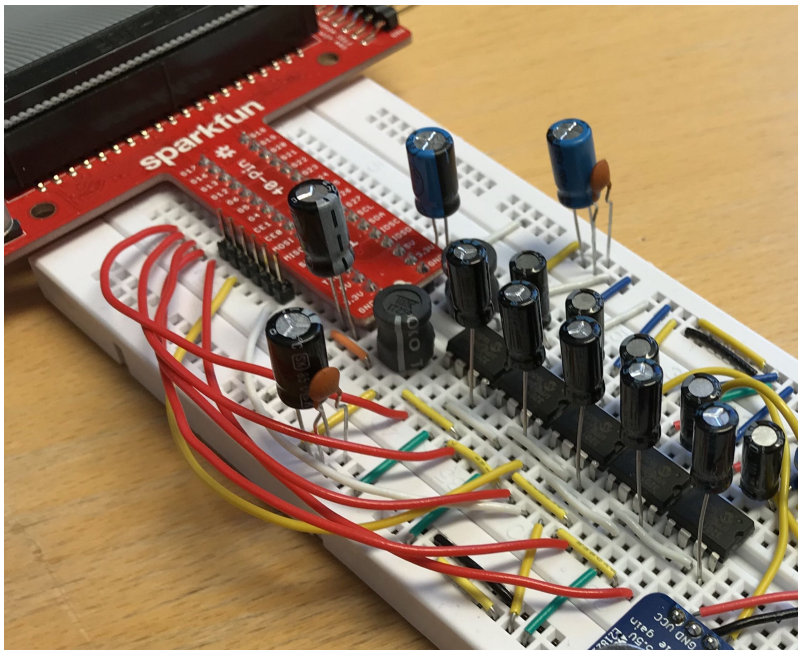


Figure 2.8: ADCs connected to the RPi with two DC-filters on the supply lines for digital and analog side. Each ADC has the same clock and chip select, separate digital outs (MISO).

2.4.3 Prepare for Sampling Data

Download the sampling program (`adc_sampler.c`) from Blackboard and put it somewhere sensible. Change the `OUTPUT_DATA` define to the file path where you want your output file to be placed, and otherwise verify the other defines at the top of the file. Verify that the pins that are defined for the various outputs and inputs actually correspond to your wired setup.

The program is compiled and linked using

```
gcc adc_sampler.c -lpigpio -lpthread -lm -o adc_sampler
```

This is almost the same as in the crash course, except that we also link the program against the libraries **pigpio**, **pthread** and the C math library. Had these been omitted, **gcc** would have complained about undefined references to central pigpio and pthread functions. A Makefile can be written to automate this linking, or you can use CMake to generate a Makefile for you. This is outside the scope of this manual, but it is usually the best practice.

Since none of the analogue input pins are connected to anything, the conversion should give some random, but relatively small numbers back as the pins are floating. Here's an example:

```
pi@elsyspi:/mnt/home/RPi/c-code$ sudo ./adc_sampler 31250
# Starting sampling: 31250 samples (expected Tp = 32 us, expected
   Fs = 31.250 kHz).
# 31250 samples in 1.127011 seconds (actual T_p = 36.064354 us,
   nominal Fs = 27.73 kHz).
# Data written to file. Program ended successfully.
```

The example output here is from an earlier iteration of `adc_sampler.c` where we got higher sample periods due to interference between the sound system and the PCM timer. This has been fixed as of 2019-02-18 by changing the timer to the PWM timer, see comment in `adc_sampler.c` at lines 117-121 (above `gpioCfgClock(5, 0, 0)`. Your `adc_sampler` should give sample periods closer to the expected sample period.

The command line argument **31250** is the number of samples to take. This is something you can vary freely.

Unfortunately, you need superuser privileges (*sudo*) to grant `pigpio` access to the HW registers. If you forget, the program will tell you.

2.4.4 Prepare adc_sampler.c for measuring signals

With the ADC up and running you should modify `adc_sampler.c` for more rigorous testing. The program is prepared for data export by writing to binary files, but with the wrong path.

If you haven't done so already, open adc_sampler.c in a text editor and change the output file path to where you want to save the data file.

Save, compile and run. Check that the files show up in your location. These will be overwritten each time you run the program.

You could also modify `adc_sampler.c` to take in the output filename as a command line argument, but we leave this up to you. [1]

For your reference again:

```
# From the directory where adc_sampler.c is located
# Compile with:
gcc -o adc_sampler adc_sampler.c -lpigpio -lpthread -lm
# And run with:
sudo ./adc_sampler 31250
```

Read and understand the code and start to fiddle around with things. This is the starting point you get, and you should develop it further to suit your needs. Always refer to the component datasheets if things are unclear. For reference to the C library, see the web page where we downloaded it from, or the repository on GitHub.

Given that the ADCs are working well, you should now proceed and look at the data in Python. Verify that the potentiometer voltage is dead stable within the LSB of error on all ADC channels. Read on in the next section to see how you can start analyzing the data with Matlab.

2.4.5 Test the ADC with a potentiometer

To verify that the ADC is actually working, meaning that it gives correct conversions, we can attach a potentiometer to the analogue inputs (or a signal generator like the lab exercise asks for). You can short all inputs together and connect them to the potentiometer all at the same time, or move the potentiometer wire from one channel to the next and run adc_sampler.c for each of them. The potentiometer should be connected between 3.3 V and ground, so that the ADC reference and potentiometer voltages are the same.

We have been asked a few times how the potentiometer works. You can think of them as 3-pin resistors. Two pins are attached to the ends of the resistor and the third is connected to the moving part where you regulate at what physical point on the resistor you extract a third voltage. It can thus be used as a simple voltage divider where you can control "both" resistances. They all look slightly different, but it should be easy to distinguish the voltage divider pin from the two end pins by inspection. You can also use the potentiometer as a variable resistor as well if you use the voltage divider pin and one of the other two pins, leaving the third pin open. Mind that when the potentiometer is turned all the way in either direction, one of the resistances will be very low! You therefore risk short circuiting a supply line if you connect the potentiometer wrong.

If you set the potentiometer to somewhere between each of its end points, you should get a value on all channels that does not differ more than one. The MCP3201 has a specified maximum conversion error of one least significant bit (LSB), i.e that the error lies in the first bit out of 12. This means that the converted value lies in range $\in [0, 4095]$ ($2^{12} = 4096$ values) with an uncertainty of ± 1. To calculate the analogue voltage from any conversion, use the relation

[1] If you run `adc_sampler` as e.g. `./adc_sampler 31250 some_adc_data_filename.bin`, `argv[1]` contains `"31250"` and `argv[2]` contains `"some_adc_data_filename.bin"`, and you'd have to use `argv[2]` instead of the `OUTPUT_PATH` define for setting `output_filename`. You'll probably want to verify that ADC sampling works before starting to fiddle around with this, however.

$$V_{\text{conv}} = \frac{C}{4095} \cdot V_{\text{ref}}, \tag{2.1}$$

where C is the numerical conversion value.

Find a multimeter and see what your V_{ref} is. It is not likely to be exactly 3.3 V. Then measure the voltage on the potentiometer that is hooked to your ADC's analogue inputs and calculate the expected value for its conversion. Does it deviate, and if so, how much? This is a task that you should do for a decent amount of voltage points in order to check that your ADCs have a linear conversion response, or at least know its limitations if it is not linear.

2.5 Data Analysis With Python

A Python script `raspi_analyze.py` and a data import function is available for download on Blackboard. The analyze script imports data from the binary file generated by `adc_sampler.c` so that you can analyze and process them. The script creates a NumPy array with data from all ADC channels along the first axis, i.e. grouped in columns.

Use `raspi_analyze.py` as a starting point for your own project. Look at the code and read the comments, both in the script and in the import function. The script will import the data from the binary and create a 2D Numpy array where each column is an ADC channel. The sample period is also retrieved from the first line of the binary.

You can easily vary the number of samples to be taken by varying the input argument to `adc_sampler`.

As a final note here, remember that NumPy operates on arrays. There are very many functions in NumPy, and most of them let you specify which axis they should operate along by passing the keyword argument `axis`, with varying default behaviour. This means that you can take the fast Fourier transform on two-dimensional arrays, and `numpy.fft.fft(x)` will handle each row in the 2-D array `x` as one dataset (it operates along the last axis by default) and transform each one separately. Thus, taking the fast Fourier transform of the **rawData** array will look like this:

```
raw_fft = np.fft.fft(raw_data) # transform the array along its rows
```

Using the nominal sample period, it should be straightforward to calculate the sample frequency, and thereafter calculate the frequency response (amplitude and/or phase) for each of the ADC channels.

If you need help on any function, type `help(func.name)` in an interactive Python environment, where `func.name` is the name of the function as you would call it in your code. Alternatively, search the documentation online.

Chapter 3

Preparation for the Acoustics Lab

3.1 Introduction

For the acoustic lab assignment, our main goal is to detect the angle at which an audio source is placed relative to our sensor. For this we will use a sensor made from three microphones in an array. The sensing in essence is a matter of measuring the relative time delay of sound between the different microphones in our array.

Here, we will go through some basics regarding the setup of our microphone array and give some hints on how we can solve the task of finding the angle of a source relative to the Raspberry Pi and microphone array by data analysis in Python.

3.2 Connecting the Microphones to the Core System

If you are reading this, we expect that you have already verified that the core system with ADCs is working correctly (lab 1). If not, you really need to follow the guidelines in the manual for lab 1 and ensure the functionality first!

3.2.1 Hooking up the Microphone Array

Start with disconnecting the potentiometer if it is still connected to the ADCs, and any other signal source left over from lab task 1. The microphone array should be arranged with the microphone elements organized as good as possible in an equilateral triangle. You have to consider the length of the triangle's sides, but it will be beneficial to have the microphones separated a few centimeters at least. In this example we'll simply make the triangle as large as we get it on a single breadboard strip. This means approximately 6.5 cm between each microphone. You should really think about what the benefit might be if you make the triangle bigger (or smaller), and what the problem might be as well.

When you have decided, lay out ground and 3.3 V lines to the correct pin placements for each microphone, and route the output signals accordingly to an available ADC.

When you have ***triple-checked*** that your wiring is correct, place the microphones on their sites, and make sure to place them ***the correct way***! The microphone's preamplifier burns immediately if you wired it wrong and put it under reverse bias.

Figure 3.1 shows two examples for how to do it.

The microphones have a variable gain that can be adjusted with a tiny regulator on the underside with a very small star-shaped screwdriver. We have prepared them to be at maximum gain, but if we missed some of them they should come with 50% gain from fabrication which also works just fine.

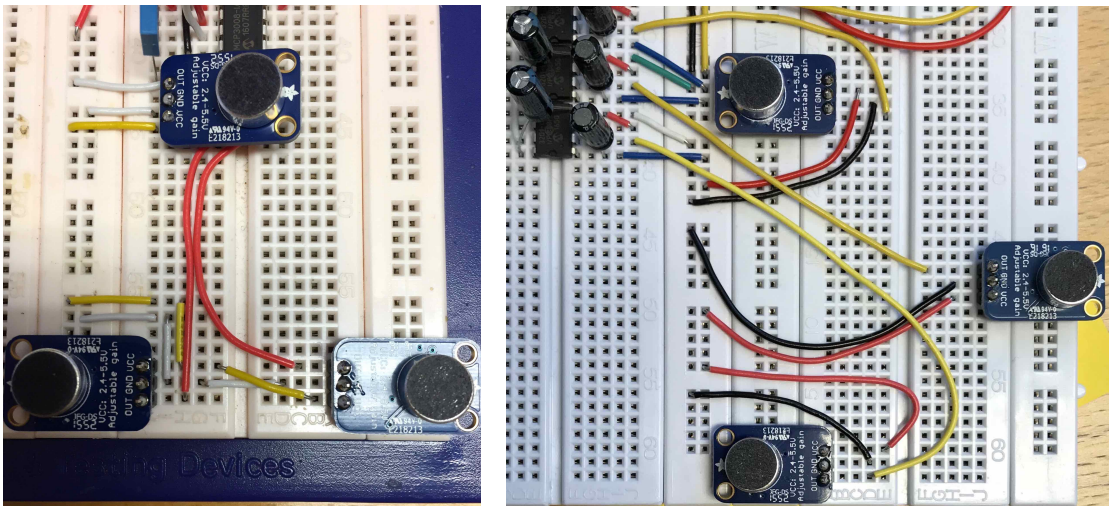


Figure 3.1: Two ways of laying out the microphone array. Left gives approximately 5.5 cm spacing and right about 6.5 cm spacing between microphones.

3.2.2 Test the Microphone Array

Again, verify that the connections are done correctly. Take note of which microphones you define as number 1, 2 and 3. This is essential for the analysis later. Use your phone with an app, Online Tone Generator 1, Online Tone Generator 2, or some other method, for generating one/more sine(s) or other signals to test the system. Use relatively low frequencies first to resolve the signals well, for example $f \in [400, 5000]$ Hz.

Read the data in with `raspi_analyze.py` (or your preferred analysis method) and look at the signals in both time and frequency domains. Consider the following points before you start with the actual lab exercises:

- Verify that you can resolve the correct frequencies (both low and high, approaching your sampling system's limit).
- Do the signals make sense in terms of relative intensity and phase?
- Is it necessary with additional filtering and/or amplification before sampling?

3.3 General Tips for the Data Analysis

Some parts of the preparatory work are major steps during your processing. Make sure to prepare the code you wrote there into general functions that you can call from your main processing routine (band pass filtering, cross-correlation, and so on). And please, comment generously so that anyone trying to understand your code can do just that.

3.3.1 Software Modifications to the Sampling Program (Optional)

The sampling program currently always writes to the same filename, `adcData.bin`.

Since you are now going to capture (multiple?) data sets, analyze them, and later also use the same sampling program for acquisition of radar signals, it could be a good idea to make the data output filename from `adc_sampler.c` an input argument instead of a hard-coded define, to avoid overwriting old data you might want to keep.

If you run `adc_sampler` as `adc_sampler 31250 filename.dat`, the string "adc_sampler" will show up in `argv[0]`, "31250" in `argv[1]` and "filename.dat" in `argv[2]` in your application. Use `argv[2]` to set `output_filename` appropriately. The variable `argc` will contain the total number of elements in the array `argv`. See otherwise the argument handling at the top of `main()`.

Remember that the programs and scripts we supply are only a starting point, and that you are encouraged to make convenient modifications yourselves.

3.4 On Supply Noise and Noise Reduction

This section is optional.

The RPi is manufactured to be as cheap as possible (obviously). This is nice for us who want to buy it, but often brings other problems. Particularly noise on analog circuitry. Since the RPi itself does not hold much analog circuitry, the circuit designers didn't care much about the analog supply lines or shielding them from all digital noise. The 3.3 V supply is bad, and the 5.0 V supply is even worse. Therefore, unfortunately, interference from network activity and other load on the RPi couples to the supply lines very easily and with relatively large swings.

To combat this noise, we have employed the DC filters and separation of digital and analog parts of our layout in lab 1. It is crucial that you continue to use enough capacitors for stabilizing and shunting if you need to build other blocks to your system.

If you want to learn more about noise reduction, bypassing and decoupling, have a look at the app-note from The Designer's Guide Community. It is also available on Blackboard. They provide other useful links to books that you might be interested in. One of them is also linked on Blackboard, for full access through the NTNU University Library.

For our purpose now, we will just live with (and neglect) this common-mode noise. As long as we make sure that our input signals to the ADCs are relatively large, the supply noise will at least not be dominating or disrupting our data completely.

Chapter 4

Preparation for the Radar Lab

The radar lab exercise will demonstrate how the Doppler shift of electromagnetic waves can be used to estimate a moving object's velocity relative to a sensor (the radar). The radar can optionally also be used to estimate the distance to an object by modulating the radar frequency (FMCW-mode), but is not a required part of the exercise.

4.1 Connecting the Radar Module to the Core System

We will use the K-LC6 v2 radar module from RFbeam. It's a small radar with two rows of eight antenna elements, one each for transmitting and receiving the electromagnetic waves. The antenna elements are laid out 1 x 8 elements, which gives us a narrow beam in the azimuth direction and a broad beam in the elevation direction. All electronics necessary to transmit and receive signals are included onboard the module so that using it is very simple.

4.1.1 Radar Module Layout

Figure 4.1 shows the pin layout of K-LC6 along with physical dimensions and what signals should be connected to each pin. **Take care not to connect anything wrong!** The radar modules are relatively expensive, and we don't have many more than we need.

The radar module has an input called $\mathrm{VCO_{in}}$ (Voltage Controlled Oscillator) which can be used to modulate the frequency of the radar wave (almost) linearly with the voltage on the $\mathrm{VCO_{in}}$. For the Doppler radar operation, we will simply leave this pin open (not in use), and the radar will default to a stable frequency at about 24.13 GHz with the VCO becoming equal to $\mathrm{V_{cc}}$ (5 V).

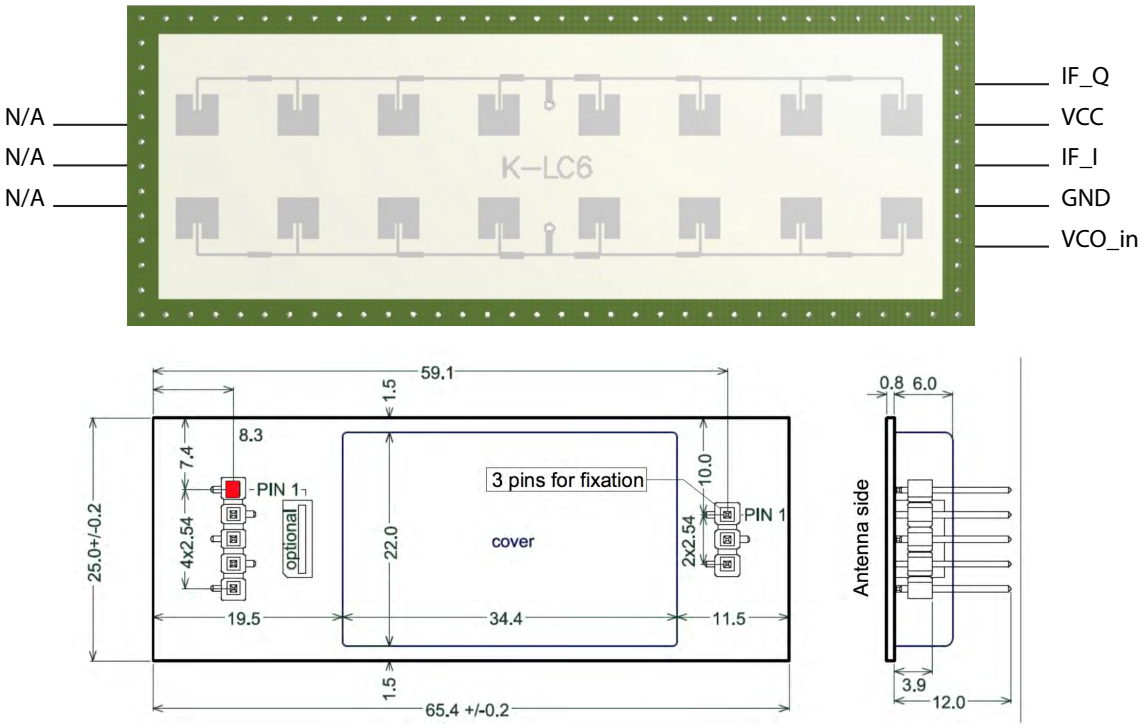
The two outputs $\mathrm{IF_I}$ and $\mathrm{IF_Q}$ are so-called *in-phase* (I) and *quadrature* (Q) signals. This means that the quadrature signal is phase shifted $\pm 90°$ relative to the in-phase signal (sign depends on direction of movement). In-phase and quadrature signals are very much used in electronic measurement systems. Check out the Wikipedia entry for initial information if you like.[1] From the radar module the in-phase and qudrature signals will be identical, though, only phase shifted to give the direction of movement of an object in range.

4.1.2 Active Bandpass Filters for the Radar Outputs

Due to relatively small reflected signals from typical objects you will use for velocity measurements, we need to amplify the output signals from the radar module a bit. In addition, since the radar module needs 5 V and our instrumentation runs on 3.3 V, we need to lower the DC offset from 2.5 V to 1.67 V.

We will use an active bandpass filter for this, built around the operational amplifier MCP602. This IC has two op-amps inside, so we'll build two active bandpass filters, one for each radar channel.

[1] For example, you can find the real and imaginary values of an unknown impedance by measuring the in-phase and quadrature amplitudes of a known AC signal applied to the unknown impedance.



Pin	Description	
1	IF output Q	
2	VCC	5V supply
3	IF output I	
4	GND	0V supply
5	VCO in	1V .. 10V, open = 5V

Figure 4.1: Radar module K-LC6 pin-layout, physical dimensions and signal ranges for the pins. Take care not to connect it the wrong way, as doing so will reverse the applied voltage and potentially burn the internal power circuit. The figures are compiled from the datasheet and modified for easier understanding, and are thus courtesy of RfBeam.

Make sure you look into the MCP602 datasheet so that you know your device, and particularly the pin layout!

In the labkit, you should find all necessary components for the active filters except the resistors needed to define the new reference (DC offset). Find some suitable ones from your inventory. Figure 4.2 shows a sketch of the schematic for one of those active filter circuits.

4.1.3 Wiring up the Active Bandpass Filters

Take time to plan a bit before you start to wire up the active band pass filters. There are quite a few components, so the breadboard area gets crowded quickly. Read the MCP602 datasheet carefully so that you connect things correctly. Check the wiring two or three times before you connect the power. Draw your own schematic based on Figure 4.2, and have a look at Figure 4.3 for inspiration.

When you have the filters ready you should test them with a signal generator and oscilloscope, to ensure that they function properly *before* you connect the radar module and ADCs at each end. Moreover, this is essential documentation for your project report.

Figure 4.4 shows the full frequency response of an active band pass filter. The gain is spot on 20 dB in the pass band, and the lower and upper cut-off frequencies are close to the specified values. In addition, you must check that the DC voltage offsets on each side of the circuits are as expected.

If your filter response looks similar to this, everything should be ready for connecting the ADCs

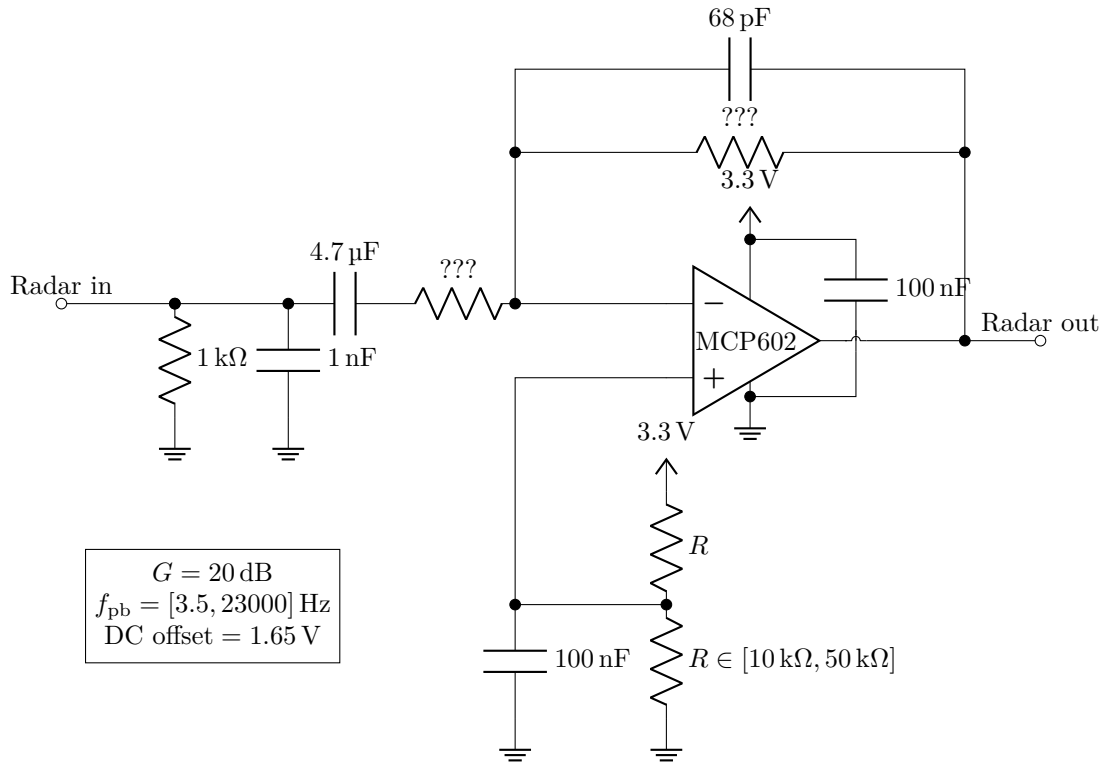


Figure 4.2: Circuit diagram of an active bandpass filter with DC offset. The in-phase or quadrature signal is used as input. The op-amp is a single supply IC with rail to rail output swing, i.e the output signal will always be in range $\in [0, 3.3]$ V, equivalent to our ADCs' measurement range.

and radar module to each end of these filters.

4.1.4 Wiring up the Radar Module

The radar module has 3 pins on the left side that are not connected to anything and merely used for support/fixation. We will not use those at all. Instead, we will use a set of headers with a resulting 90° angle to get the radar module to stand upright.

Route the 5 V and GND signals to pins 2 and 4, respectively. The 5 V supply from the RPi is dreadful and couples a lot of noise from the digital side on the RPi. Therefore, make sure you bypass well with both large stabilizing capacitors ($\approx 100\,\mu$F) and smaller shunting capacitors (≈ 0.1 nF to 100 nF). This will not remove the noise, but as long as our signals are relatively large it should be okay. If you experience really bad issues with the supply, try an external 5 V supply and common ground.

In the example circuit/photo of the example circuit, an inductor is also included on the 5 V bypass in order to emulate the low-pass filter you built during lab 1 for the 3.3 V supply. This is optional, and something you can try to improve your circuit further.

Continue with connecting the in-phase output signal (pin 3), $\mathrm{IF_I}$, and the quadrature output signal (pin 1), $\mathrm{IF_Q}$, to each channel of the active circuit. Then connect the active circuit outputs to ADC channels 4 and 5. See Figure 4.3 for inspiration.

When you have triple-checked that your wiring is correct, mount the radar module carefully into the breadboard with the 90° header mount. Make sure you hit the right lines, and that you have it turned the correct way (we don't want to reverse bias it!).

The mounted radar module should look something like Figure 4.5, where you can see the complete system with the microphones and radar together, connected to the ADCs.

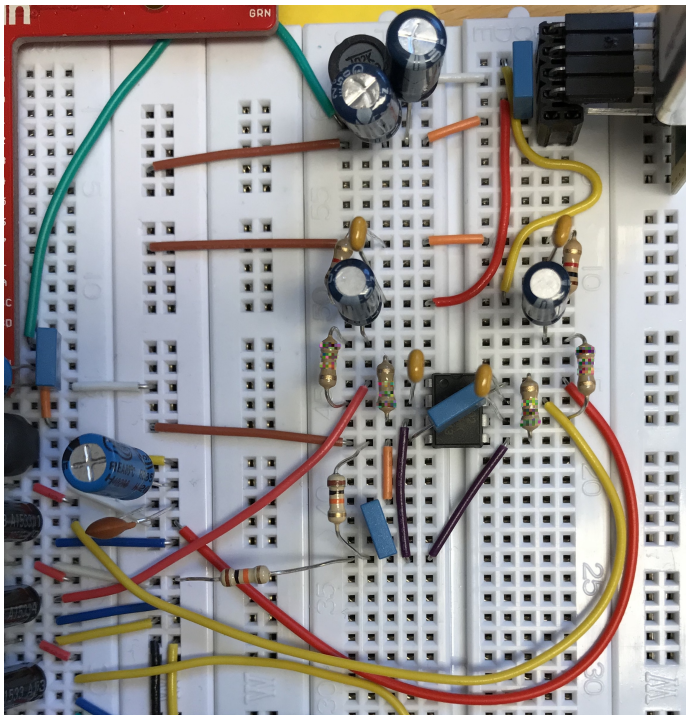


Figure 4.3: Example of two parallel active band pass filters, one for each radar signal channel.

4.2 Testing the Radar Module

You are now ready to test the radar. With an oscilloscope connected to the ADC inputs in parallel, you can see the signals live by moving a hand or book or something in front of the radar.

Also do some test runs with the sampling program. If you have a metallic reflector, use this, if not, try with your hand or a book or something. Try to move the item with a stable velocity away from or towards the radar module while sampling. Look at the data in Python. You should detect two sinusoidal signals that are phase shifted $\pm 90°$ depending on the direction you move the item. Next step is to write Python code for FFT to analyze the Doppler shift in the frequency domain. Remember to use a complex FFT based on $\mathrm{IF_I}$, and the quadrature output signal, $\mathrm{IF_Q}$

$$S(f) = IFFT(IF_\mathrm{I} + \mathrm{j}IF_\mathrm{Q}) \tag{4.1}$$

When using a complex signal, the spectrum should be asymmetric showing either positive or negative Doppler shift.

Hint: Try to multiply the raw signal with different window functions (Hanning, Hamming, Kaiser etc) before taking the IFFT to study how they will affect the sidelobes of the spectrum. This is best viewed when plotting the spectrum using dB. It is also important to remove the DC-offset before taking the IFFT. (Try with and without removing the DC-offset to see the difference)

The frequency of this signal can be used to estimate the velocity according to the preparatory exercises.

4.3 Optional: Use the Radar Module in FMCW-mode

The radar carrier wave frequency can be modulated with the $\mathrm{VCO_{in}}$ pin. In this mode, the distance to a reflecting object can be estimated. FMCW stands for *Frequency Modulated Continuous Wave*. To use the radar in FMCW-mode, attach a signal generator to the $\mathrm{VCO_{in}}$, give it a common ground, and make sure the modulation signal is in range $\in [1, 10]$ V. See the data sheet for details.

A 20 Hz triangular or sawtooth signal is nice to get some ramping time within each modulation period. If you hold some object at a stable distance from the radar, the Fourier transform of each

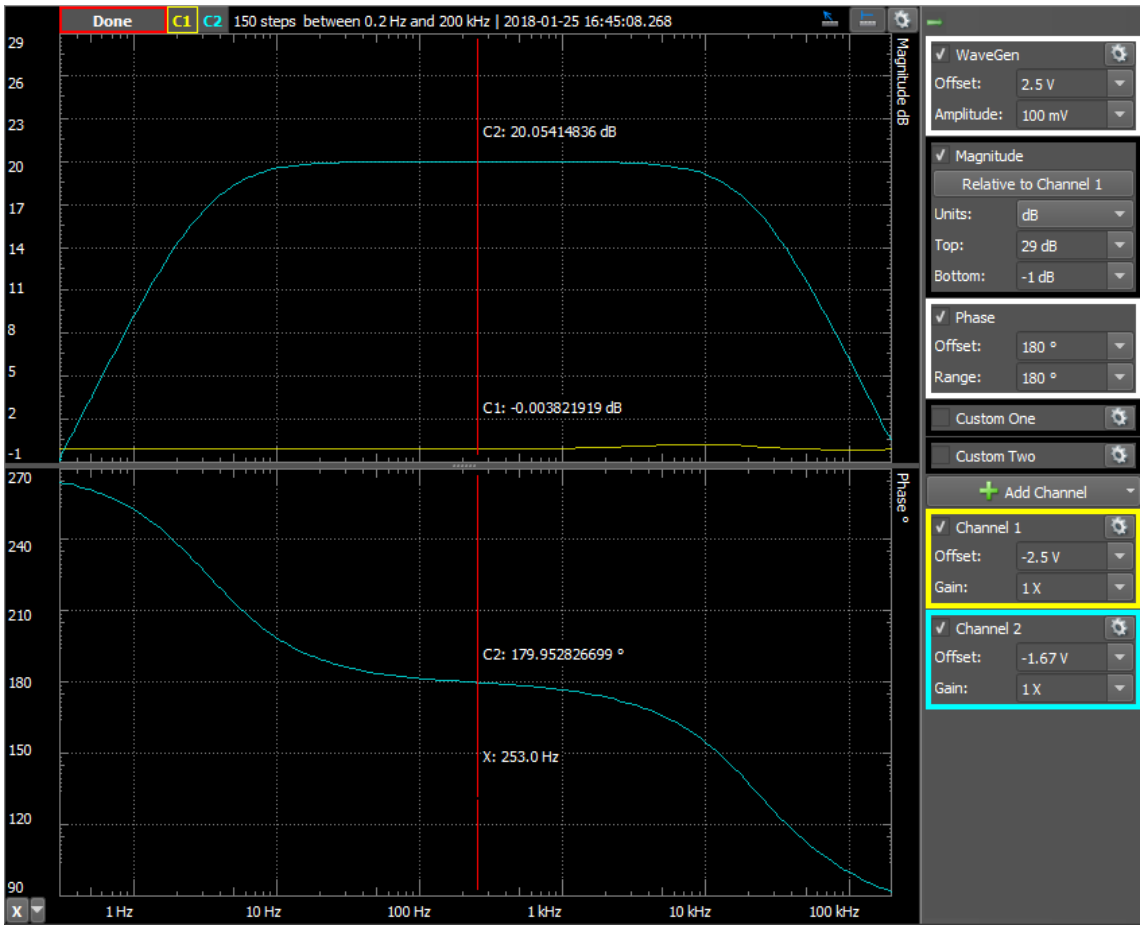


Figure 4.4: Frequency response of the active band pass filter (blue) relative to the input signal (yellow). Note that the phase response is inverted (180°)since we use the inverting amplifier mode for the op-amp circuit.

period, excluding the spikes at the ends of each ramp, should give you a frequency component that is proportional to the distance, the so-called beat frequency signal.

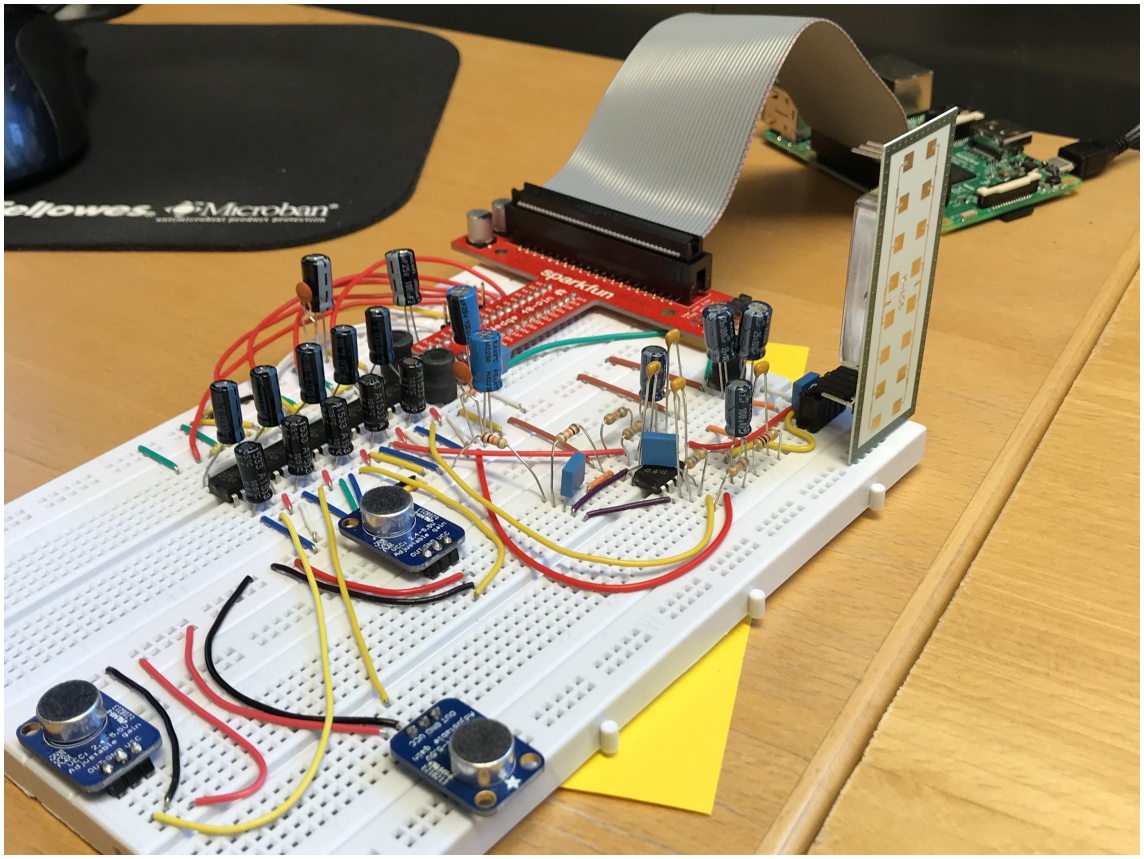


Figure 4.5: Complete system after lab sessions I–III. You can now locate the direction of a sound source, and its radial velocity if it is within the radar's angular range, at the same time.

Chapter 5

Preparation for the Optics Lab

5.1 Introduction

The Optical Lab exercise will demonstrate how we can use an ordinary low-end consumer camera to detect a person's heart rate, both by reflected light from the skin and via transmitted light through a finger or similar. For descriptions on the lab exercise, refer to the lab text. Below, we will go through the steps necessary to the get the camera up and running.

5.2 Raspberry Pi Camera Setup

5.2.1 Connecting to the Camera Serial Interface (CSI)

The RPi has a special contact for the picamera:

1. Pull the white handle carefully up and towards the Ethernet port.
2. Insert the ribbon cable with the blue side facing the Ethernet port.
3. Push the white handle carefully down.

See figure 5.1. See also the very first lab lecture done at the beginning of the course, uploaded to BlackBoard, and https://www.raspberrypi.org/documentation/configuration/camera.md.

5.2.2 Software setup

After having connected the camera to the RPi board, first enable the camera by running

```
sudo raspi-config
```

and then selecting 'Interface Options', 'Camera' and choose to enable it. It might also be that you have to add yourself to the "video" group in order to give your user enough privileges to access the video input, which is done by running

```
sudo gpasswd -a [username] video
```

where you insert the username of your main user on the system instead of '[username]'. This is probably not necessary if you still use the default user "pi". During camera enabling, raspi-config edits a config file which is read only during boot, so we have to reboot. Reboot the RPi, e.g. by typing `sudo reboot`.

After reboot, the camera should be ready. Try to acquire an image and a video using

```
raspistill -v -o test.jpg
raspivid -v -o test.h264
```

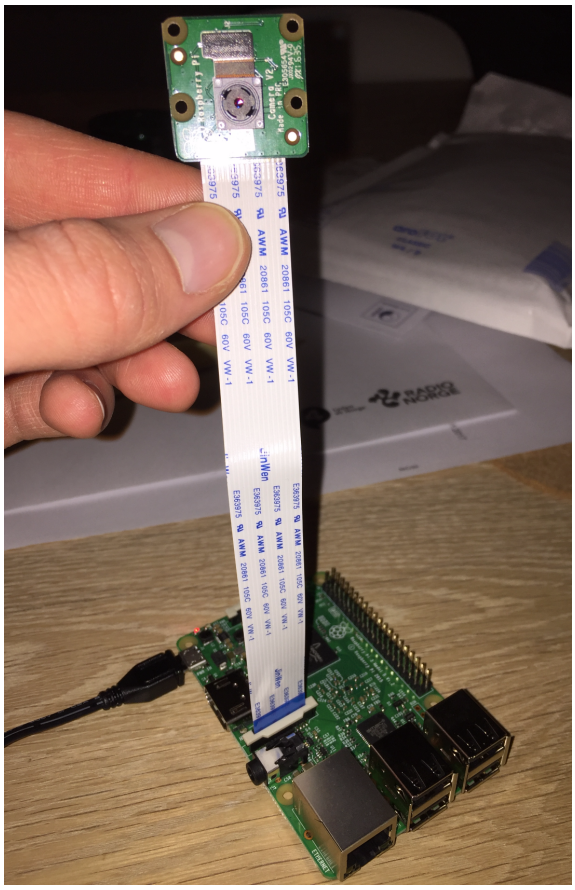


Figure 5.1: The picamera hooked up to an RPi.

Verify that there were no errors, and that the produced image makes sense. In order to properly play the video, see section 5.4.2.

A couple of software libraries are necessary for the video acquisition script that you will use later. Install them using:

```
sudo apt install python-picamera python3-picamera gpac
```

Python-picamera is the Python module for accessing the camera, while gpac is a package which includes a tool for wrapping video files in MP4 containers. If `apt` is not found, try `apt-get`.

5.3 Preparations for the Lab

We have provided you with a Python script that takes care of the video recording and conversion of the video file to a signal that can be processed.

- record_video.py: Record a video.
- read_video_and_extract_roi.py: Read the video and extract the pulse signal as the mean over each frame in a smaller region of interest.

The Python video acquisition script must be accessible from the Pi.

Read the comments in the video acquisition script to see what is being done there and instructions for how to run it. Identify parameters that might need to be changed during acquisition.

5.3.1 Some notes

- Make sure that the region being measured has sufficient amount of light, and also that the light does not cause specular reflection and white spots in the video.
- Changing the ISO value has caused some problems after the h264 is converted into mp4, so it is recommended using the default `iso=10` from the Python script and rather change the `awb_gains` value. You might need different `awb_gains` values for different points you want to measure on.

5.4 Information about the video recording scripts

5.4.1 The Picamera Library

It is possible to access and use the camera in different ways, but for this lab we recommend that you use the Picamera Python library. The library allows us to control the camera with the simplicity and readability that Python offers. If you want to experiment a bit yourself (which is recommended) have a look at the Picamera documentation here. Here you will find basic examples explaining how to capture regular images and regular videos, along with some more advanced examples when you feel ready for it.

In addition to checking out some examples to familiarize yourselves with the library we recommend you to check out this. Here we can see the supported resolutions we can choose from, and at what frame rates we can capture video in at the different resolutions.

5.4.2 Video Codec

A video codec is a device or software which has the purpose of compressing or decompressing a digital media file. We will use a codec called *h264*. This codec does not usually contain information about frame rate, length and such. If you try to play the .h264 file directly in VLC or a another video player, notice that it will not be able to display the length of the video, and depending on the frame rate chosen for the recording in your Python program it will probably be played at a faster or slower rate than what you expected. The reason for this is that it cannot find information about the frame rate in the .h264 file, so it will play it using a default frame rate. Thus, we need to wrap it in a container so that we can process the information afterwards. A container can contain different type of media files. If you have a video file encoded with the h264 standard, an audio mp3 file and a srt subtitle file, we can put all of these in a container so that we have a video with audio and subtitles. The container format we will use is the `mp4` format.

We use the pre-installed FFmpeg library to do the conversion from a file `input.h264` captured at a frame rate 40 to *output.mp4*:

```
ffmpeg -framerate 40 -i input.h264 -c copy output.mp4
```

This is already done in the example Python file we have provided for you, called **record_video.py** on Blackboard.

Now, if you try opening the new mp4 file in VLC again you will see that we have information about the time, and the frame rate it is played at should be correct.

5.5 On-Board Processing (optional)

After you are finished with the implementation of your pulse detection algorithm, have acquired the required data, and you are well satisfied with the results, you might also want to try implementing your algorithms on-board the RPi and try to estimate the pulse in real-time, or just in Python on the RPi.

See the acoustic lab text for general tips for processing in Python. We use the same techniques in the optical lab: cross-correlation and band pass filtering. Finding maxima might be a bit more difficult. Here, you could use something called smoothing splines to fit the noisy autocorrelation

using smooth third degree polynomials, and use built-in functionality for estimating the derivatives and the roots of the derivatives.

For real-time processing, this will probably get a bit more involved. You might need to do this in a sliding window-way: Decide on a number of samples you need to collect in order to be able to estimate a pulse. Wait until you have collected this number of samples. Estimate the pulse. Then move your window one sample with the time direction, and continue until finish, possibly let the window skip a couple of more samples depending on the computational requirements of the processing.

PiCamera has support for writing to a circular ring buffer (http://picamera.readthedocs.io/en/release-1.12/api_streams.html). This could be used to process data in parallell with the video acquisition. This could be combined with OpenCV, which has image processing routines which could be used for object tracking, face detection, etc.