

User Guide for the Laboratory

TTT4280 - Sensors and Instrumentation

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. However, there are far more students than assistants; please make an effort to solve your problem before contacting them, as most issues are quick fixes which can be solved by, for example, reaching out to a fellow student instead.



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

Contents

| | |
|---|-----------|
| 1 Raspberry Pi | 4 |
| 1.1 Introduction | 4 |
| 1.2 Installing the OS and Network connection | 4 |
| 1.2.1 Wireless connection | 5 |
| 1.2.2 Ethernet connection | 5 |
| 1.2.3 Connecting to a router | 5 |
| 1.3 SSH | 5 |
| 1.3.1 Enable SSH on the Raspberry | 6 |
| 1.3.2 Connecting to the Pi | 6 |
| 1.3.3 Finding the IP address | 6 |
| 1.4 Change the hostname | 7 |
| 1.4.1 Change the password | 8 |
| 1.5 File transfer using SFTP | 8 |
| 1.6 Create a simple Python script | 8 |
| 1.7 Further reading | 9 |
| 1.8 Legacy way of installing Raspbian | 9 |
| 1.8.1 Downloading the OS | 9 |
| 1.8.2 Writing the ISO image to the SD card | 9 |
| 2 ADC | 10 |
| 2.1 Introduction | 10 |
| 2.2 Raspberry Pi GPIO Pins | 10 |
| 2.3 The Pi Wedge Breakout Board | 10 |
| 2.4 ADC Control | 11 |
| 2.4.1 Low Level Hardware Register Control | 12 |
| 2.5 Instrumentation Circuit | 13 |
| 2.5.1 Connecting the Supply Line Noise Reduction Filter | 13 |
| 2.5.2 Connecting the ADCs | 14 |
| 2.5.3 Prepare for Sampling Data | 15 |
| 2.5.4 Test the ADC | 16 |
| 2.6 Data Analysis With Python | 16 |
| 3 Acoustics Lab | 17 |
| 3.1 Introduction | 17 |
| 3.2 Connecting the Microphones to the Core System | 17 |
| 3.2.1 Hooking up the Microphone Array | 17 |
| 3.2.2 Test the Microphone Array | 18 |
| 3.3 General Tips for the Data Analysis | 18 |
| 3.3.1 Software Modifications to the Sampling Program (Optional) | 18 |
| 3.4 On Supply Noise and Noise Reduction | 19 |
| 4 Radar Lab | 20 |
| 4.1 Connecting the Radar Module to the Core System | 20 |
| 4.1.1 Radar Module Layout | 20 |
| 4.1.2 Active Bandpass Filters for the Radar Outputs | 20 |
| 4.1.3 Wiring up and testing the Active Bandpass Filters | 21 |

| | | |
|----------|---|-----------|
| 4.1.4 | Wiring up the Radar Module | 22 |
| 4.2 | Testing the Radar Module | 23 |
| 5 | Optics Lab | 25 |
| 5.1 | Introduction | 25 |
| 5.2 | Raspberry Pi Camera Setup | 25 |
| 5.2.1 | Connecting to the Camera Serial Interface (CSI) | 25 |
| 5.2.2 | Software setup | 25 |
| 5.3 | Preparations for the Lab | 27 |
| 5.3.1 | Some notes | 27 |
| 5.4 | Information about the video recording scripts | 27 |
| 5.4.1 | The Picamera Library | 27 |
| 5.4.2 | Video Codec | 27 |
| 5.5 | On-Board Processing (optional) | 28 |

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 and Network connection

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](#). This course uses Raspberry Pi 3B+, if you for some reason are using another model, refer to the official documentation; nonetheless, make sure you select the correct Raspberry Pi device type. Select the 32-bit version and your SD card.

Without any peripherals such as a screen or keyboard, we need a working network connection to be able to communicate with our Raspberry, so before moving on, we recommend setting up the network connection from Raspberry Pi Imager.

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 all. For the lab sessions, we suggest that you use your phone as a hotspot.

This approach is the safest, and we recommend it. There are other approaches, which you can find in Section 1.8.

*Be **extremely** careful when selecting the target device for flashing. This will overwrite existing memory on the target device such that it is lost forever. Your operating system may just allow you to overwrite your entire hard drive, even if the OS is running from said drive.*

1.2.1 Wireless connection

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 from [the official documentation](#).

You can set up `ssh` (remote access), `hostname` and the `network connection` from Raspberry Pi Imager's Customisation tab. Remember the password you've set and enable `ssh`.

If you have trouble connecting to or confirming you have connected to your Raspberry Pi device, see [1.3.1](#). Note that your Raspberry Pi may only support 2.4GHz networks (these are more common as of 2025), but may not support 5GHz network connections, which are becoming more widespread. Some smart-phone hotspot settings may now default to 5GHz.

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

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.

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.

1.2.2 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 changing 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.

1.2.3 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. After connecting to the Pi using `ssh` (see [1.3.1](#)), you may also configure it to connect to *multiple* networks by entering

```
nmtui
```

into the command line, which will open a GUI for doing so.

1.3 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.

1.3.1 Enable SSH on the Raspberry

You can skip this subsection if you used the Raspberry Pi Imager and enabled SSH in the customisation settings. If you have not tried doing so, please reconsider doing it.

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. If you did not enable SSH when you used the Imager, you can ask the Pi to enable it 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

1.3.2 Connecting to the Pi

Insert your SD-card, power up the Pi and 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 the following command in the command line.

```
ssh username@hostname.local
```

Here, `hostname` is the hostname of the Pi. The default hostname is *raspberrypi* if you did not change it in the Raspberry Pi imager. `username` is the username you chose when you flashed the SD card. The default username is `pi`. The `.local` works much like normal toplevel domain, such as `.com` or `.no`, but tells your computer to search 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 by peeking at your terminal window. Confirm with enter.

The Pi may have trouble connecting to the hotspot of some newer iPhone models. If you experience this, enable *Best compatibility* in the *Internet sharing* settings.

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 an mDNS client. After finding the IP-address, simply connect using

```
ssh username@ip-address
```

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

For more info, check the official [documentation](#).

1.3.3 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.

Scanning the network with nmap (recommended)

nmap is a versatile and handy tool that simply checks a range of IP addresses 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 with your preferred package manager, such as `apt`.

¹Windows 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.

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 hotspot 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.

1.4 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. The hostname can be configured from the Imager, and the hostname is written to the file `/etc/hostname`. If you wish to change the hostname, edit it with

```
nano /etc/hostname
```

in the terminal. You may need to be root to save the new hostname, in which case you write `sudo nano /etc/hostname`. The Pi needs to be rebooted before these settings take effect.

“Valid characters for hostnames are ASCII(7) letters from **a** to **z**, the digits from **0** to **9**, and the hyphen (-). A hostname may not start with a hyphen.”

— Manual page for `hostname(7)`

The default hostname is `raspberrypi`. Examples of valid, and more personalised, hostnames are, `bragesimenjam`, `rodust-maskin`, `icanhazpi`, `barbenheimer2023` or `lv100miraidon`.

1.4.1 Change the password

Setting up the Pi with default credentials and `ssh` enabled is a major security risk. To change the password of the current user, use `passwd` while connected to the Pi.

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

1.5 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 your 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.

You can also mount the Pi as a drive or network device, letting you access files as though they were on the file system on your computer. The procedure depends on your operating system, and will not be covered in detail here.

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 MacOS, 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/<username>/`. This is *your* folder and you are free to create folders and files here.

1.6 Create a simple Python script

Experienced Linux/RPi users can skip this section.

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.

1.7 Further reading

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

- [Linux Command Line Cheat Sheet](#)

1.8 Legacy way of installing Raspbian

This section is for those who dislike the convenience of using an imager.

1.8.1 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.

1.8.2 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.

Do not boot the device before doing the steps in Section 1.2.1.

Chapter 2

Connecting the ADC

2.1 Introduction

The instrumentation setup we will use is based on three analog-to-digital converters (ADC) of type MCP3201. The MCP3201 is a single-channel ADC with 12 bits resolution, and we will run 3 units in parallel for sampling three channels simultaneously. Three channels will be used for microphones in the acoustics lab. For the radar lab, two of the channels will be used for the Doppler radar. In addition, we need other circuitry like filters and amplifiers for noise reduction, signal filtering and/or amplification.

This manual should provide most of the information you need 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.

We expect you to use 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 signal 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 pick 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 noisy signals into our circuit (because of their transient nature), and should be kept away from analog circuitry as best one can. 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.

2.2 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.

2.3 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](#).

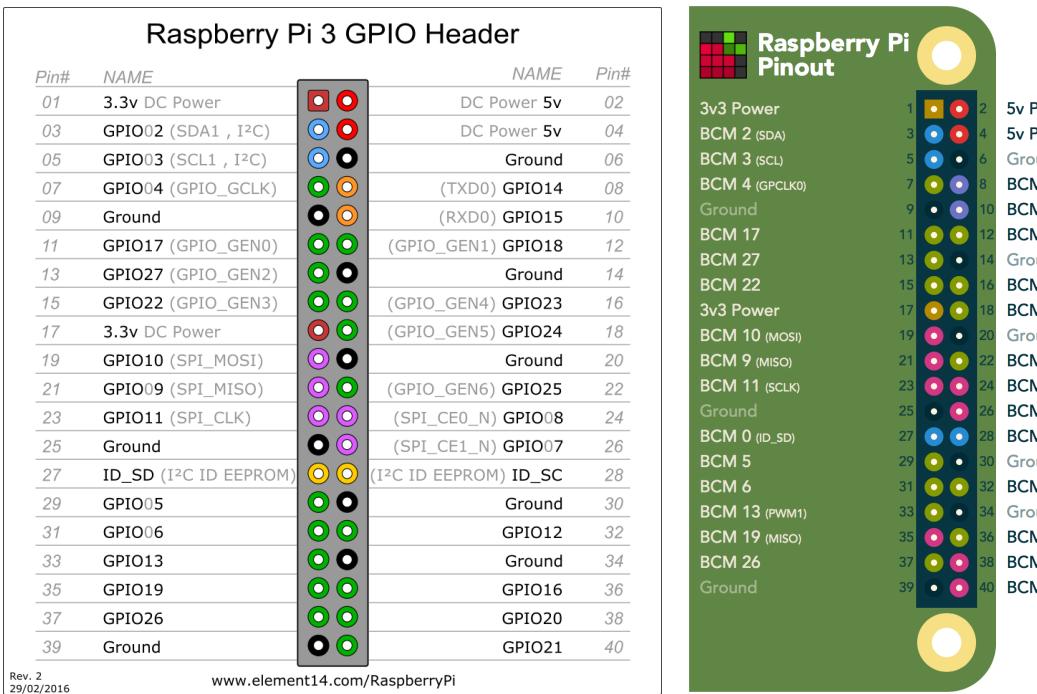


Figure 2.1: Two variants of the Raspberry Pi 3B 40-pin header. GPIO or BCM (chip pin names) and hardware module names are listed alongside the respective physical pins. Figure courtesy of [Farnell/Element14](#) (left) and [Raspberry Pi Pinout](#). (right). Check out the latter one, lots of graphical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

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 (except the radar which runs on 5 V) to avoid complications with the logic levels of the RPi GPIO pins.

2.4 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 modules, located in the periphery of the actual CPU, sample and control the pins without interference

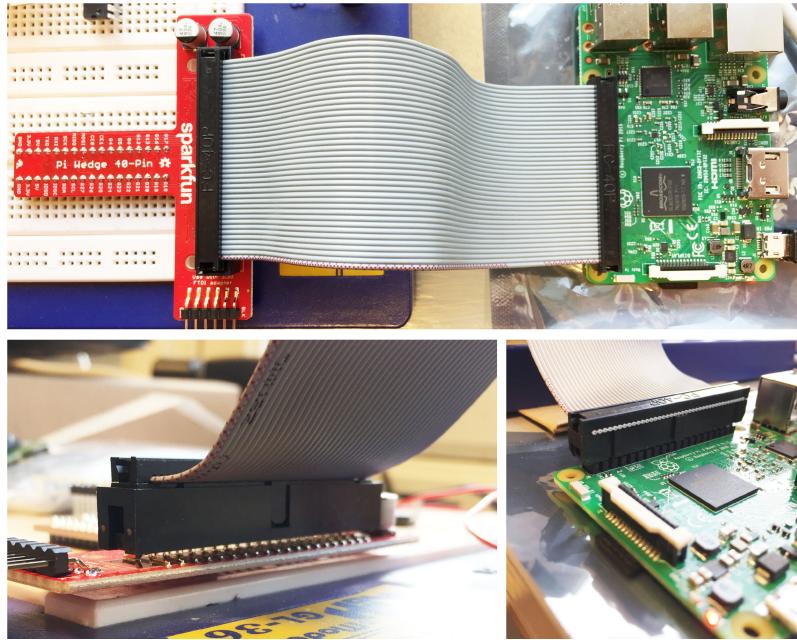


Figure 2.2: How to connect the Pi Wedge breakout board.

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.

2.4.1 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 almost all set for sampling on the RPi side. We thus move on to the circuitry.

2.5 Instrumentation Circuit

The instrumentation is simple, using three ADCs in parallel. They should use a common wire/signal for the clock and chip select signals, and 3 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 choose your own inputs for signals to the Pi, the wires labelled "DATA(1-5)".

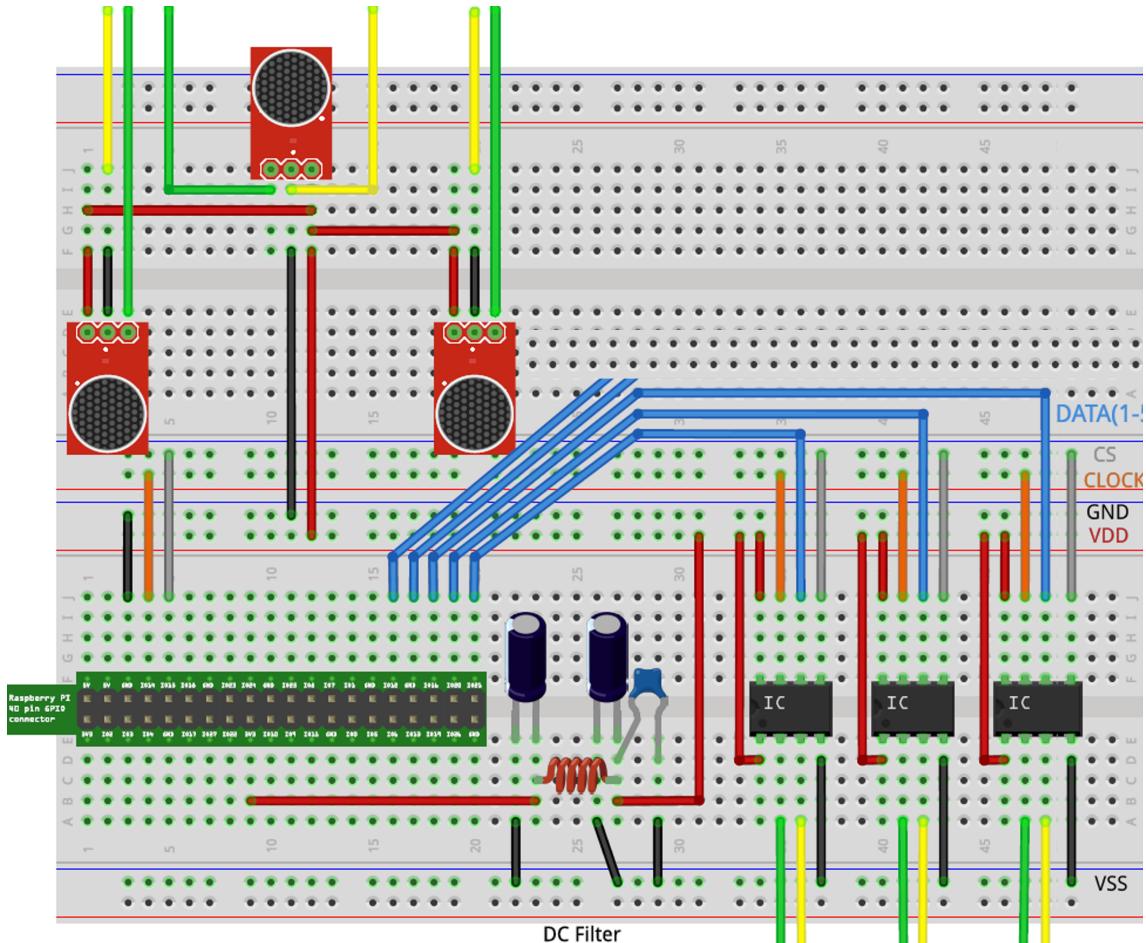


Figure 2.3: The system as a whole should look something like this, picture does not include decoupling capacitors.

2.5.1 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 an LC-filter low-pass filter, or DC-filter, on the 3.3 V supply line for the analog circuits, illustrated in Figure 2.4. This filter consists of a large 100 μ F stabilizing capacitor, an inductor (drossel) and a combination of a large and small capacitor. Please search literature for LC low-pass filters to find suitable theory for calculating cut-off frequency, damping etc.

Wire up this filter close to the analog 3.3 V supply and make sure that all ground pins are connected to 0V. Place the capacitors on the right side in the supply rails along the sides of the breadboard.

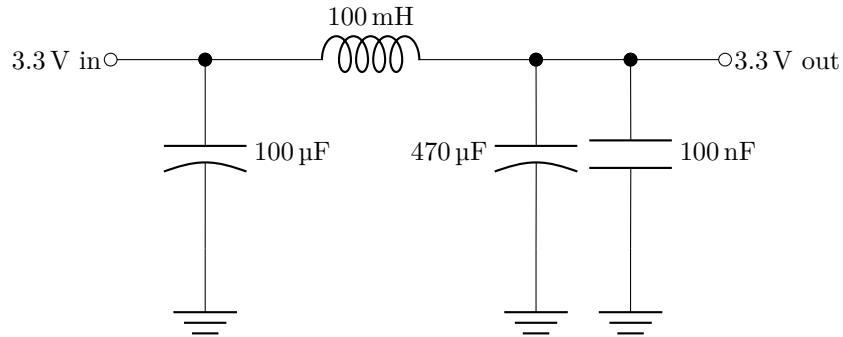


Figure 2.4: Schematic sketch of a supply line noise reduction filter.

Verify that the filter works by measuring the frequency response of the filter (Bode plot, 1 Hz - 100 kHz) using the Digilent without connecting the filter to the ADCs. (*Hint: Use 100 mV amplitude for the measurement. The Digilent has problems driving higher amplitudes for these kind of filters*). Remember to export the Bode plot data to CSV file – taking a screenshot of the measurement in Waveforms is not appreciated in a report. Discuss

This should ensure that we at least stabilize the analog supply network properly and reduce the noise to an acceptable level.

2.5.2 Connecting the ADCs

Check 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 3 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 sequentially, so leave all analogue inputs open at first.

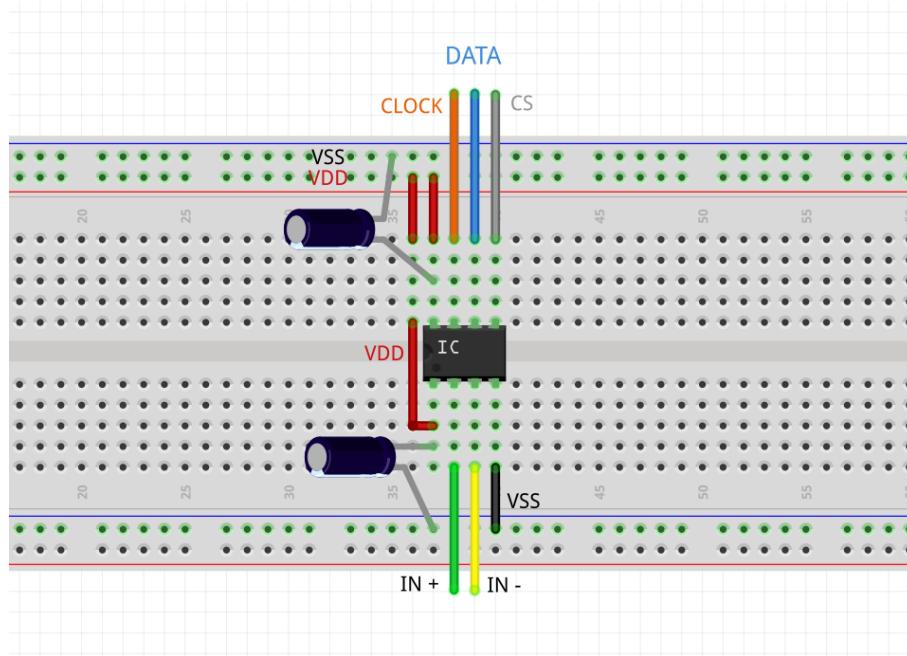


Figure 2.5: Model of the connection setup. Each ADC needs its own decoupling (bypass) capacitor for the power supply line.

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.

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} pin.

Note that both the 3.3V supply and the V_{ref} need decoupling capacitors. Add the 1 μ F decoupling capacitors directly between all V_{dd} and V_{ss} pins, and the 1 μ F 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!*

2.5.3 Prepare for Sampling Data

Download the sampling program (`adc_sampler.c`) from Blackboard and put it somewhere sensible.

Usage: `sudo ./adc_sampler <sample-count> [output]`

`sample-count` is required, and specifies the number of samples to take from the ADCs. `output` is optional, and the sampler writes the ADC readings to a time-stamped file if omitted. If you wish to override this behaviour, you can redefine `OUTPUT_DATA` 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 numbers correspond to GPIO pin numbers, as Listing 2.1 shows.

Listing 2.1: Excerpt from the head of the ADC sampler, containing the pin number definitions.

```

41 /* RPi PIN ASSIGNMENTS */
42 #define MIS01 25      // ADC 1 MISO (GPIO pin number 25)
43 #define MIS02 12      // ADC 2 MISO (GPIO pin number 12)
44 #define MIS03 16      // ADC 3 MISO (GPIO pin number 16)
45 #define MIS04 20      // ADC 4 MISO (GPIO pin number 20)
46 #define MIS05 21      // ADC 5 MISO (GPIO pin number 21)
47
48 #define MOSI 10       // GPIO for SPI MOSI (GPIO 10 aka SPI_MOSI)[...]
49 #define SPI_SS 24     // GPIO for slave select (GPIO 24).
50 #define CLK 23        // GPIO for SPI clock (GPIO 23).
51 /* END RPi PIN ASSIGNMENTS */

```

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.

Remember to recompile every time you change something in the C-code.

If you did not connect the inputs, 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.

Unfortunately, you need superuser privileges (*sudo*) to grant `pigpio` access to the HW registers. The sampler fails if you do, and asks you to run it as root.

2.5.4 Test the ADC

To verify that the ADC is actually working, meaning that it gives correct conversions, apply a waveform from the signal generator to the inputs. You can apply it to one ADC at a time and sample many times, or all in parallel and sample once. Most important is that the ADCs are fed with the same signal, so avoid voltage-divider solutions. The MCP3201 has a specified maximum conversion error of one least significant bit (LSB), so that the error lies in the smallest bit. 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

$$V_{\text{conv}} = \frac{C}{4095} V_{\text{ref}}, \quad (2.1)$$

where C is the numerical conversion value, found in the preparation tasks.

2.6 Data Analysis With Python

A Python script `raspi_import.py` is available for download on Blackboard. It provides a function for importing 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. Check the docstring for example usage.

As a final note here: NumPy functions operate on arrays. Mathematical functions often apply element-wise. Functions that calculate statistics, such as `numpy.sum`, `numpy.var`, often apply to all elements in the *flattened* array, and return a scalar. In this case, providing the `axis` keyword argument instructs the function to apply to all elements along the specified axis only, and return an array.

```
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 build a correct frequency axis for power spectral density (PSD) estimates.

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. If your measurements give signals with different gain, check that the regulator is turned 90° from the notch to ensure that they are set to maximum gain.

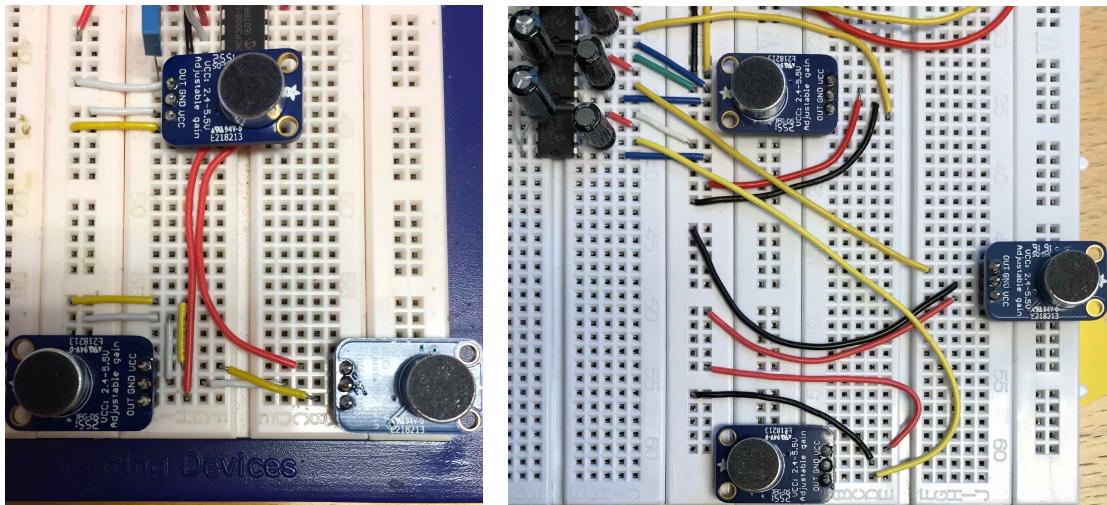


Figure 3.1: Two ways of laying out the microphone array. Left gives approximately 5.5 cm spacing and right about 6.5 cm spacing between microphones.

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 filter for the 3.3V analog supply line. It is crucial that you continue to use bypass capacitors close to the V_{cc} pins of the ADCs.

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 24 GHz radar module from [RFbeam](#). It is a small radar with two rows (arrays) of eight antenna elements, one row for transmitting and one row receiving the electromagnetic waves. The antenna array is laid out with 1×8 elements, which gives us a narrow beam in the azimuth direction and a broad beam in the elevation direction. Read the datasheet to figure out which plane has the narrowest beam (i.e. what is the azimuth and elevation directions for the radar). All electronics necessary to transmit and receive signals are included onboard the module.

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 VCO_{in} (Voltage Controlled Oscillator) which can be used to modulate the frequency of the radar wave (almost) linearly with the voltage on the 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.

The K-LC6 needs 5V DC to operate. We recommend to use the 5V output from a Digilent kit to supply the radar since the 5V output from Rpi is not stable or powerful enough to drive the radar.

The two outputs IF_I and IF_Q are so-called *in-phase* (I) and *quadrature* (Q) signals. This means that the quadrature signal is phase shifted $\pm 90^\circ$ relative to the in-phase signal (sign depends on object's 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 quadrature 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, we need to amplify the output I- and Q- signals from the radar module. We use an op-amp which is configured as an active bandpass filter for this purpose. The bandpass filter should have a low-frequency cut-off to remove any

¹For 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.

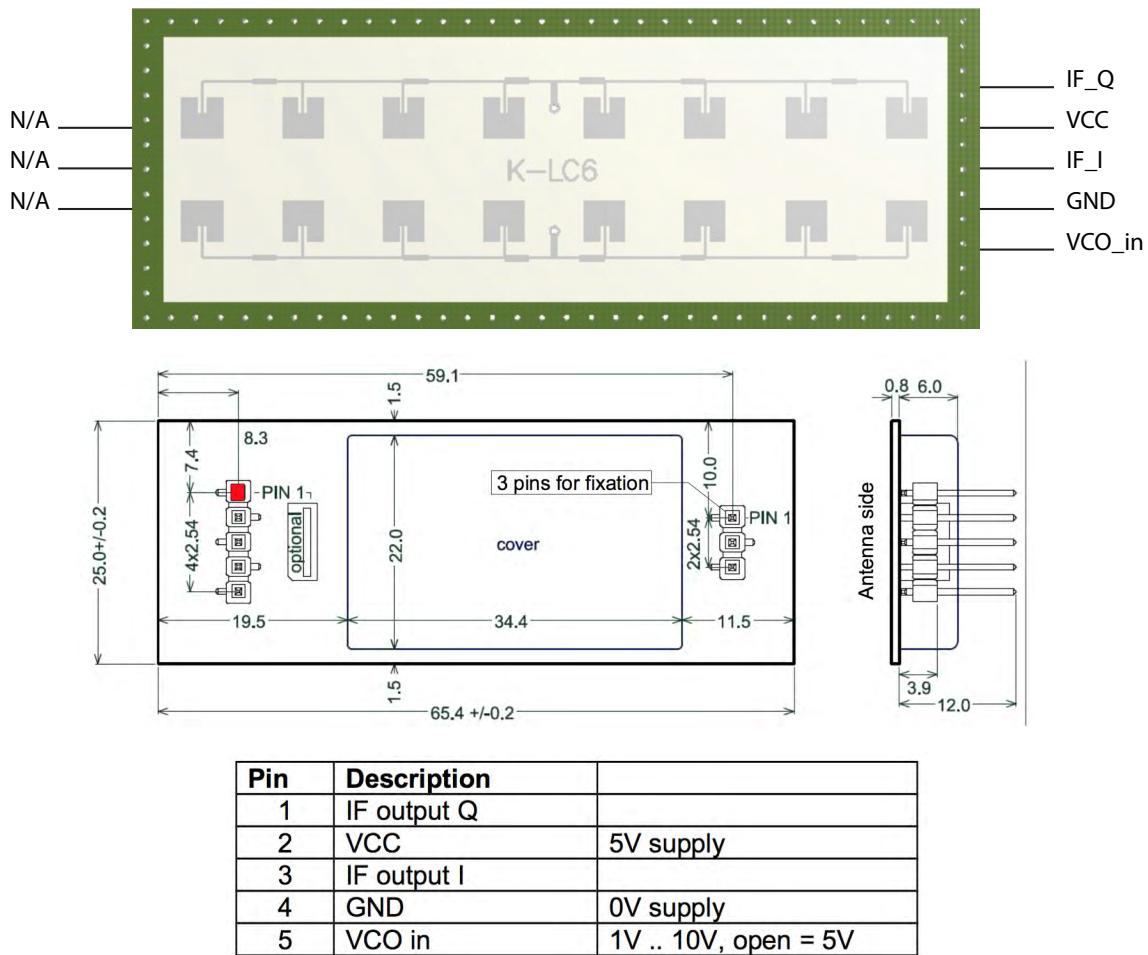


Figure 4.1: Radar module K-LC6 pin-layout, physical dimensions and signal ranges for the pins. Take care not to connect it the wrong way, as doing so will reverse the applied voltage and potentially burn the internal power circuit. The figures are compiled from the datasheet and modified for easier understanding,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

DC-offset from the I- and Q- signals, and a high-frequency cut-off to suppress high frequencies that are outside the range of interest. In addition, since the radar module needs 5 V and our ADC runs off 3.3 V, we need to lower the DC offset of the I- and Q-signals from 2.5 V to 1.67 V to match with the ADC input voltage range.

The active bandpass filter is built around the operational amplifier MCP6002. This IC has two op-amps inside allowing us to build two active bandpass filters, one for each radar channel. Make sure you study the MCP6002 datasheet so that you know your device, and particularly the pin layout!

In the lab kit, you should find all necessary components for the active filters except the resistors and capacitors needed to define the new reference (DC offset). Figure 4.2 shows a sketch of the schematic for one of those active filter circuits.

4.1.3 Wiring up and testing 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 MCP6002 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 verify that they function properly *before* you connect the radar module and ADCs. **This is**

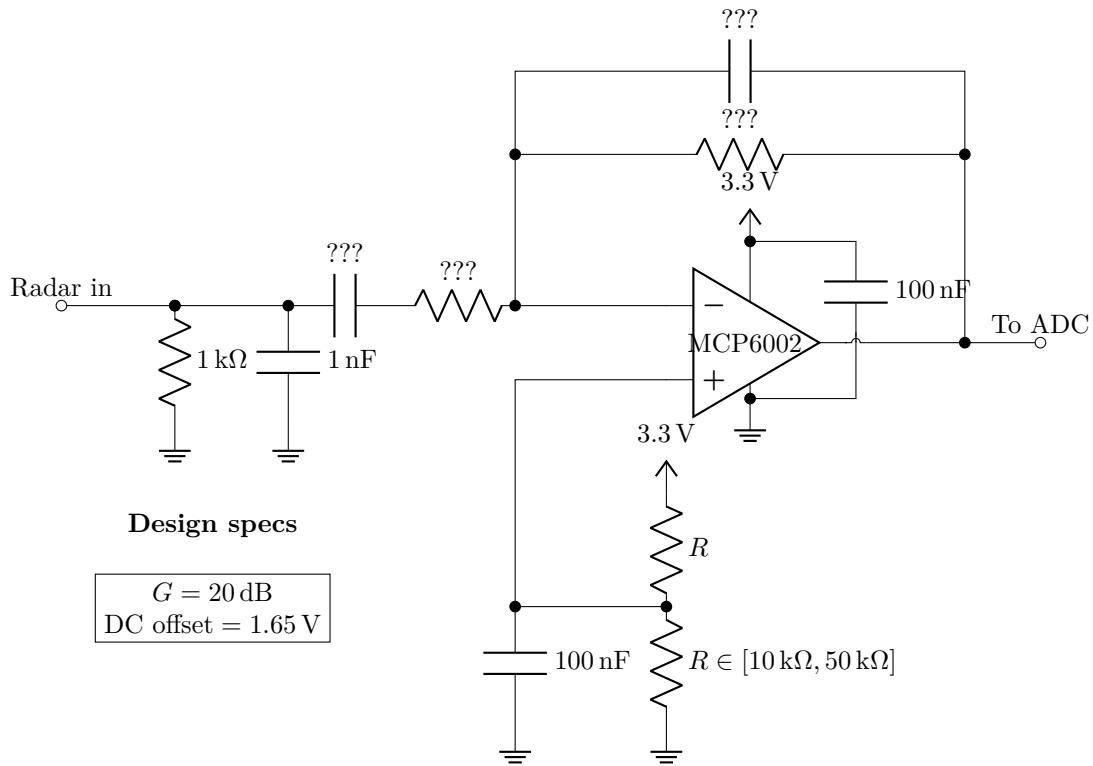


Figure 4.2: Circuit diagram of an active bandpass filter with DC offset. The in-phase or quadrature signal is used as input. The op-amp is a single supply IC with rail to rail output swing, i.e the output signal will always be in range $u \in [0, 3.3] \text{ V}$, equivalent to our ADCs' measurement range.

essential documentation for your project report.

Use the Digilent kit to measure the frequency response of the active bandpass filters. The gain should be 20 dB in the pass band, and the lower and upper cut-off frequencies should be 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 ok, everything should be ready for connecting the radar module to the input and the ADCs to the output.

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. Use an external 5V supply from a Digilent kit instead of the 5V output from the Rpi.

Continue with connecting the in-phase output signal (pin 3), IF_I , and the quadrature output signal (pin 1), IF_Q , to each channel of the active bandpass filters. Then connect the filter outputs to ADC channels 4 and 5. See Figure 4.3 for inspiration. Take a little time to think whether the radar now has the narrowest beamwidth in the vertical or in the horizontal plane.

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 4.4, where you can see the complete system with the microphones and radar together, connected to the ADCs.

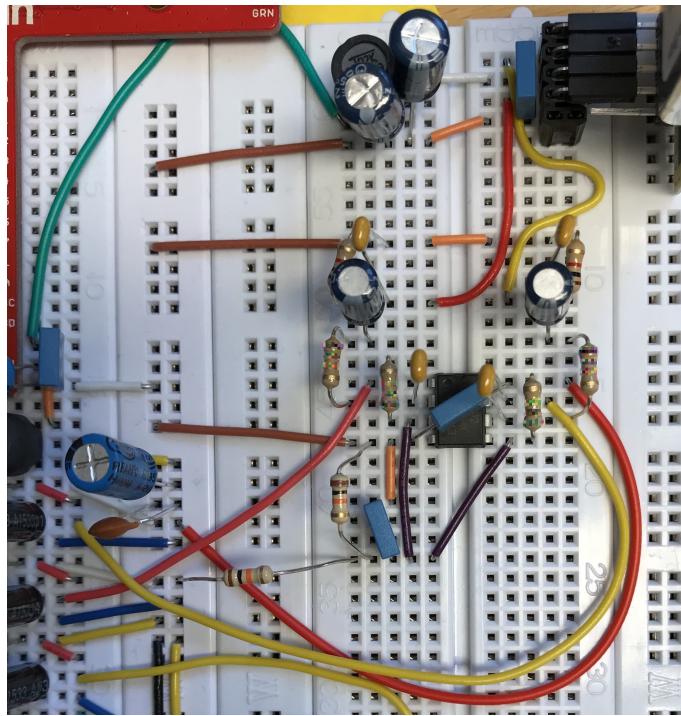


Figure 4.3: Example of two parallel active band pass filters, one for each radar signal channel.

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. Note that the Doppler shift will only be produced when the object is moved radially towards or away from the radar.

Also do some test runs with the sampling program. Try to move the object 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 $\pm 90^\circ$ 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 combining the IF_I and the quadrature output signal, IF_Q , into a complex signal.

$$S(f) = FFT(IF_I + j * IF_Q) \quad (4.1)$$

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

Hint: Try to multiply the raw signal with different window functions (Hanning, Hamming, Kaiser etc) before taking the FFT 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 FFT. (Try with and without removing the DC-offset to see the difference.)

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

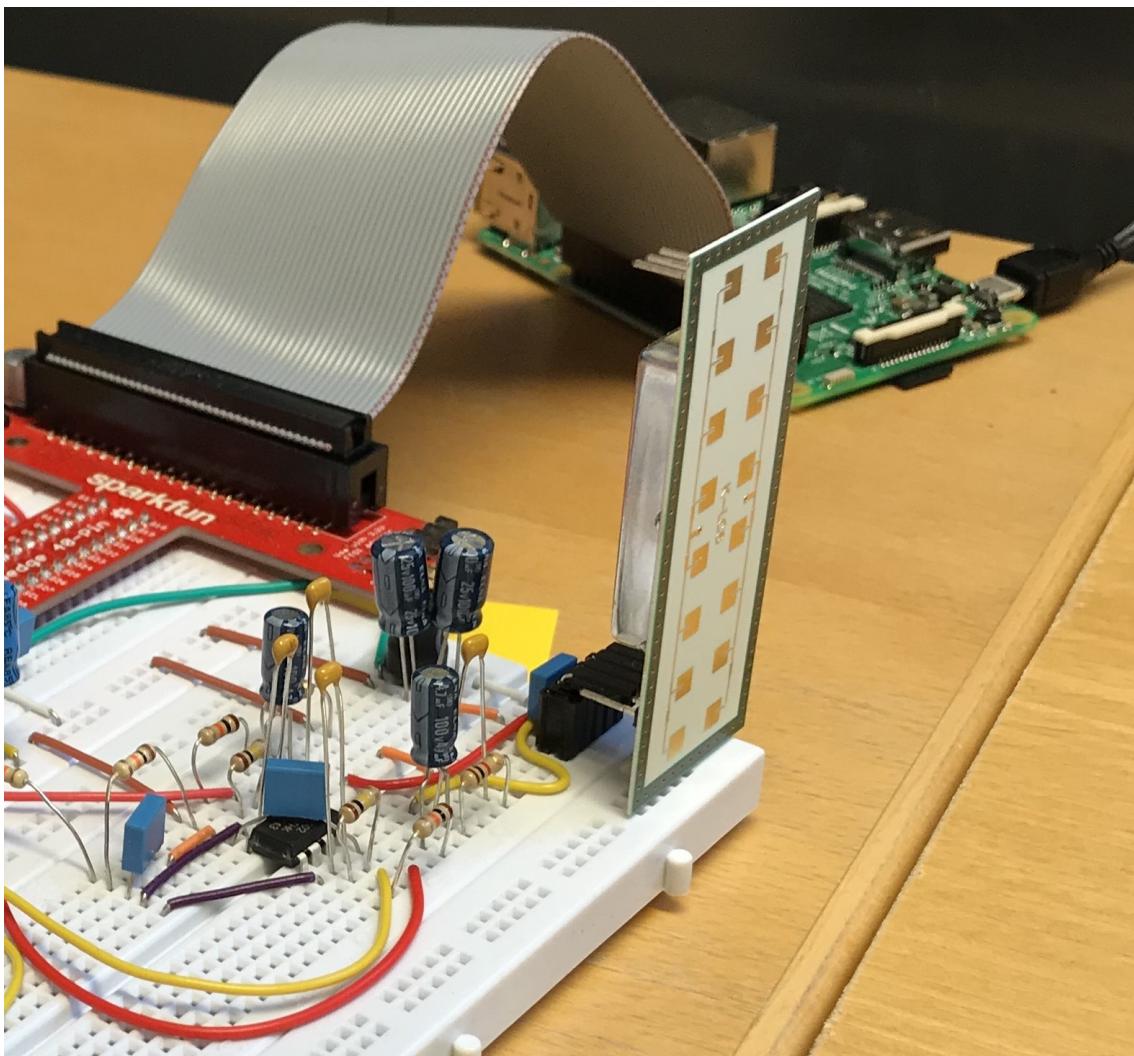


Figure 4.4: Radar and bandpassfilter installed on the breadboard

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 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 connector 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

The instructions in this box concern the Buster version of Raspbian. The legacy camera module is deprecated on Bullseye, and is not available on Bookworm.

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.

Starting with Bullseye, the Raspberry Pi camera is already activated. Verify that the camera is accessible by executing

```
rpicam-hello
```

on the Pi (requires `ssh -Y` and an environment that supports the X window system). Substitute `rpicam` by `libcamera` if you are on Bullseye.

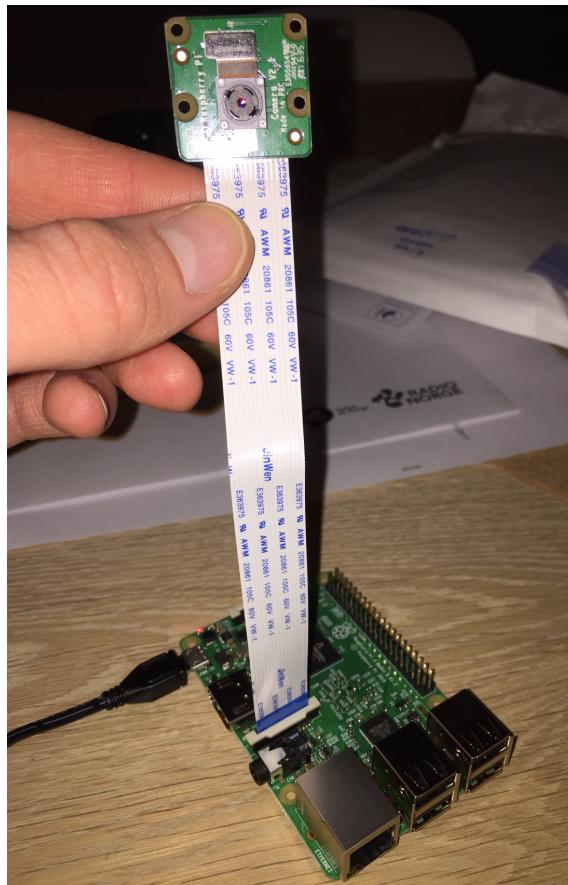


Figure 5.1: The picamera hooked up to an RPi.

If you ssh-ed to the Pi without the `-Y` flag, you can test camera functionality by executing

```
rpicam-still -v -o foo.jpg
rpicam-vid -v -o foo.h264
```

instead, changing `rpicam` to `libcamera` if on Bullseye rather than Bookworm. Note that `sudo` privileges may be required.

The `raspistill` and `raspivid` applications are used on Buster. They are legacy and will not work on later versions of the OS.

Consult [this page](#) for more details on `rpicam` applications.

Verify that there were no errors, and that any produced images or videos make 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. The default installation should already have these `apt` packages installed, but if not, you can install them using

```
sudo apt install python-picamera2 python3-picamera2 gpac
```

If the `picamera2` modules are unavailable, which could be the case if you are on Bullseye or earlier, install the `picamera` modules instead.

`python-picamera2` is the Python module for accessing the camera, and `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.

`record_video_upgrade.py` Record a video, with the `picamera2` module.

`read_video_from_roi.py` Choose a spatial slice of the video, and find the average of each colour channel in the region of interest for each frame.

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 `picamera2` 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 – and we recommend this approach – have a look at the `picamera2` 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 parallel 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.