ESEIAAT

Final Bachelor Thesis

UNIVERSITAT POLIÈCNICA DE CATALUNYA

DEGREE IN INDUSTRIAL AND AUTOMATIC ELECTRONIC ENGINEERING

Upgrade of automation system in the wind tunnel of Fluid Mechanics

TR4

Student:
Dylan Beck

Supervisors:
Prof. Robert Castilla,
Prof. Gustavo Raush

Delivery date: 22/06/2021

Contents

Abstract

Upgrade of automation system in the wind tunnel of Fluid Mechanics, Dylan Beck, Universitat Politècnica de Catalunya. Abstract of Bachelor's Thesis, Submitted 22 June 2021:

The aim of this thesis is to repair and update the old wind tunnel project at the UPC campus Terrassa. The wind tunnel is an old project that is used for aerodynamic studies at the university. The tunnel can simulate different scenarios and measure the effects on the objects placed in the tunnel. But the installation is not used in a while due to a defective Y- and Z-axis for repositioning the sensor. The hardware and software used is also outdated and needs an update. For this update some parts need to be replaced. The slow PC with the old software needs to be replaced and the stepper motors powering the Y- and Z-axis need a new driver. Research for new components, software and coding is needed to update this project.

The decisions made had the aim of creating a flexible installation with many opportunities for the future. By using a new Raspberry Pi as controller coded with Python, future students can find lots of information online due to the big community around it. Many possibilities for future extensions are created by using the I2C protocol that allows easy addition of new devices.

This Final project or Trabajo de Fin de Grau (TFG) for my master degree has been executed at the Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual (ESEIAAT) campus de Terrassa, a part of the Universitat Politecnica de Catalunya (UPC). The TFG was a part of the Erasmus+ program offered by the Bachelor of Energy Technology at the Odisee University College in Ghent Belgium.

List of Figures

List of Tables

Table of abbreviations

I2C	Inter-Integrated Circuit
UART	Universal Asynchronous Receiver/Transmitter
SPI	Serial Peripheral Interface
SCL	Serial clock
SDA	Serial Data
ACK	Acknowledge
Pi	Raspberry Pi
OS	Operating system
TFG	Trabajo de Fin de Grau
ESEIAAT	Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual
UPC	Universitat Politecnica de Catalunya

1 State of the art

The wind tunnel is an old project from the 90's located at the UPC Terrassa TR4 building. It's purpose is simulate airflow in different situations and measure the pressure it applies on the object inside. Than students can study the impact on the model placed in the tunnel. In the tunnel there are two different measurements. One for the pressure on the object to see the effects of the airflow on it. The other measures the speed of the airflow at different points in the tunnel. These two measures are controlled by an old PC running on an old version of Linux. The stepper motors who move the axes are controlled by a 3.5A EVO 2 controller, connected to the PC. Because of the old hardware, software and a defect at the air speed measuring part, the project needs an update white some new components and software. The PC hardware is very old and slow and the software can use an update to. The pressure measuring still works fine, but the airflow measuring doesn't work properly anymore.
The project is located at the TR5 building of the UPC Terrassa campus. It is used for the courses about aerodynamic.

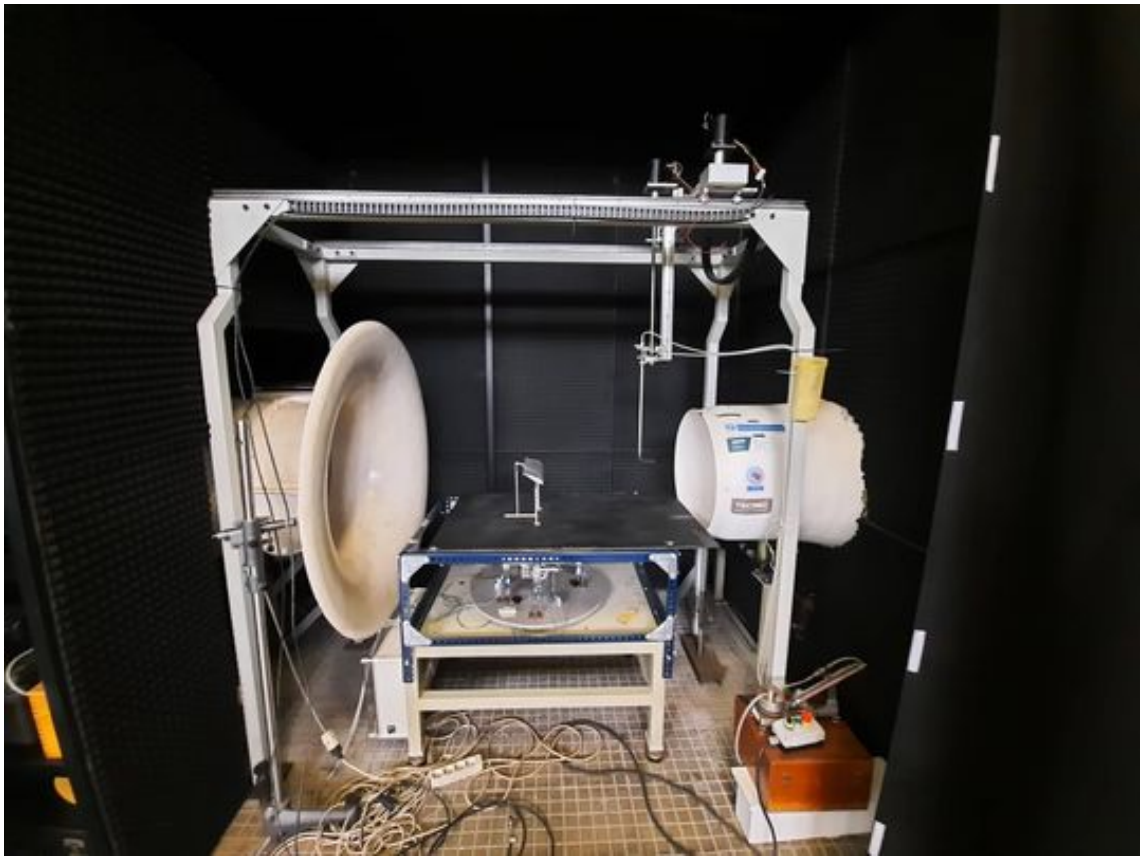

Figure 1: Wind tunnel project

In the picture we see the inside of the wind tunnel. The air flows from right to left trough the big tubes. In the middle is a model of the wings of a plane located. It will be positioned in the middle of the air stream and the pressure applied on the wings gets measured by the system below it. In the top right there are two motors who drive the Y- and Z-axis of the sensor, with the sensor attached underneath. The X-axis is mover manually, there is no amortisation attached to this movement.

2 Update proposal

For this project I will update the controller of the tunnel and repair the Y- and Z- axis. Research for new components is needed to replace the outdated ones. The communication between all the components is very important and will also be analysed.

The PC used is very old and slow. A new Raspberry Pi 4 with monitor and the Raspberry Pi operating system (OS) will replaces the PC and will also control the stepper motors. As communication protocol, I2C is suggested for its advantages. With I2C bus the quantity of wires will be reduced and the system will be very flexible for adding new parts in the future. The old motor drive will be replaced by a new one that supports I2C communication.

Only the parts controlling the airflow meter and the PC will be reworked. Other parts still work fine and won't need a replacement. So will the pressure measurement stay the same and the power supply can be reused to.

3 Stepper motors

Here we are going to go over the specifications of the stepper motors used in the project. This are two RS PRO Hybrid Stepper Motors (Figure 2).

Figure 2: RS 440-470 stepper motor

3.1 working

A Stepping motor is a brushless DC motor. The motor rotates by performing steps. His full rotation is divided in equal steps. The stepper motor can take all possible positions allowed by the number of steps and hold this position when powered. This feature is obtained by the structure of the motor. It allows to know the exact angular position of the rotor by simply counting how may steps have been performed.
Like every electrical motor, a stepper motor has a static part (stator) and a dynamic/rotating part (rotor). By energizing a stator phase, a magnetic field is generated by the current flowing in the coil and the magnets in the rotor will align with this field. By powering multiple phases in order, the rotor will rotate or hold his position at a certain angles. In Figure 3 we see a example of the basic working with a angle of 60°. [15]

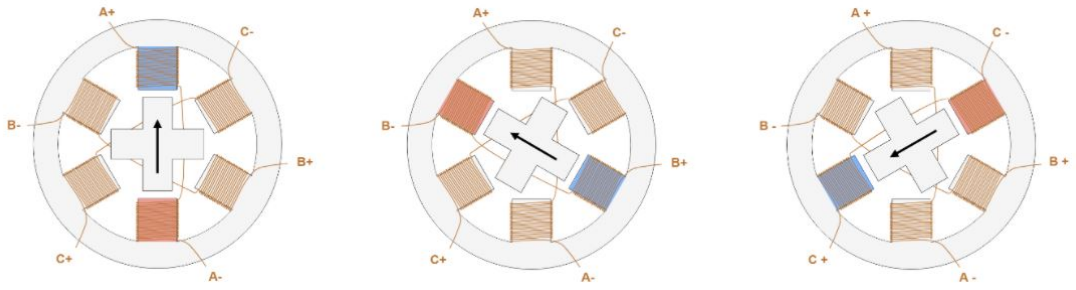

Figure 3: stepper motor steps

3.1.1 microstepping

With microstepping the amount of steps can be multiplied. A disadvantage is that the maximum speed will decrease the same rate as the increase in steps. This because no controller can send out an infinite number of steps.
The functioning of microstepping goes as follows. The basic working stays the same, but we decrease the angle of a step. By decreasing one coils power while increasing the next coils current, the shaft will slightly turn to the second coil. The smaller the current differences applied to the coils, the smaller the steps. This can go from half steps to 1/10 of a step to even smaller. Depending on the driver u use. [15] [14]

3.1.2 Hybrid stepper motor

The motors used in the wind tunnel are hybrid stepper motors. The inside looks like Figure 4.

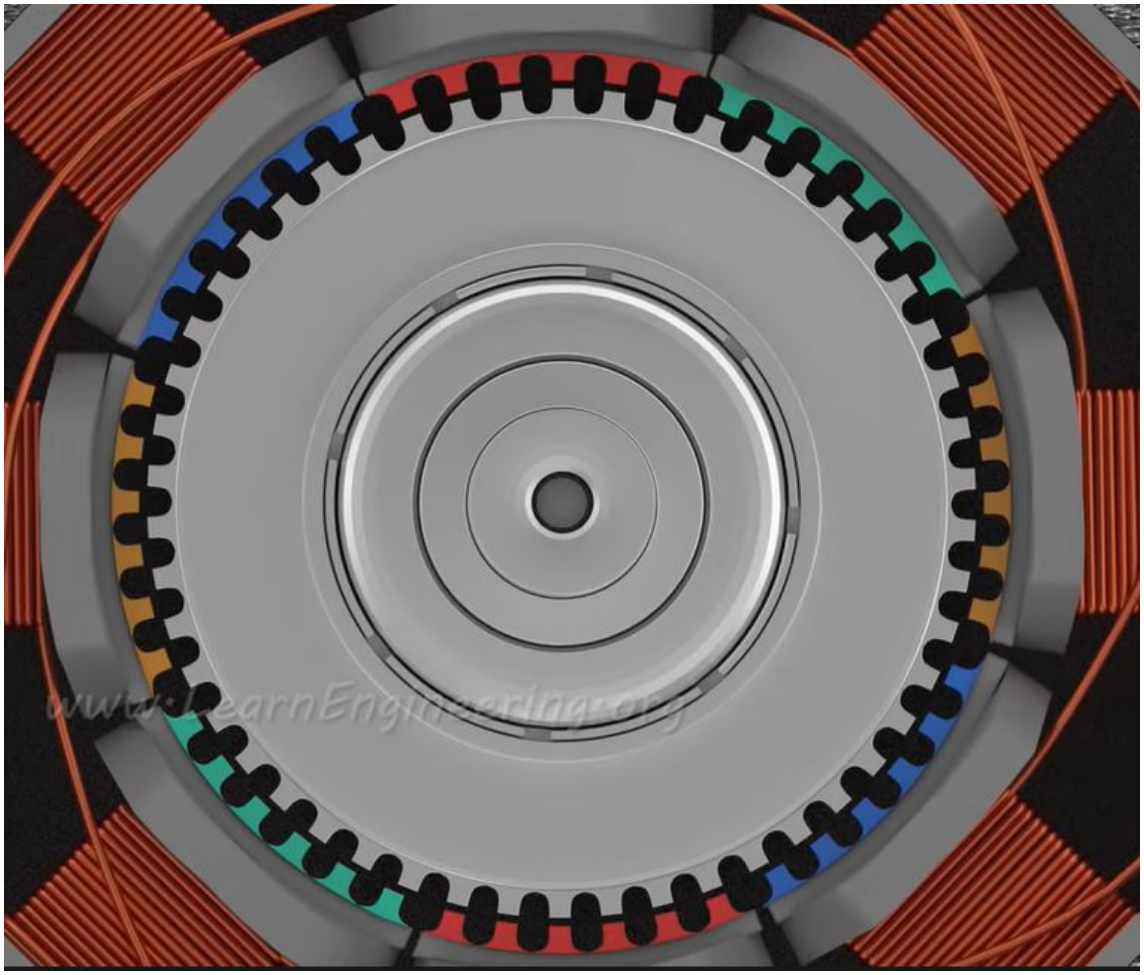

Figure 4: Structure hybrid stepper motor

The rotor and stator both have teeth. The rotor 50 and 48 on the stator. The stator has two less teeth because they are aligned different than the ones of the rotor. The stator is divided in four groups of teeth, on Figure 4 you can see the different groups by the four different colors. There are always two groups half aligned with the teeth of the rotor, one group is fully aligned and one is not aligned.

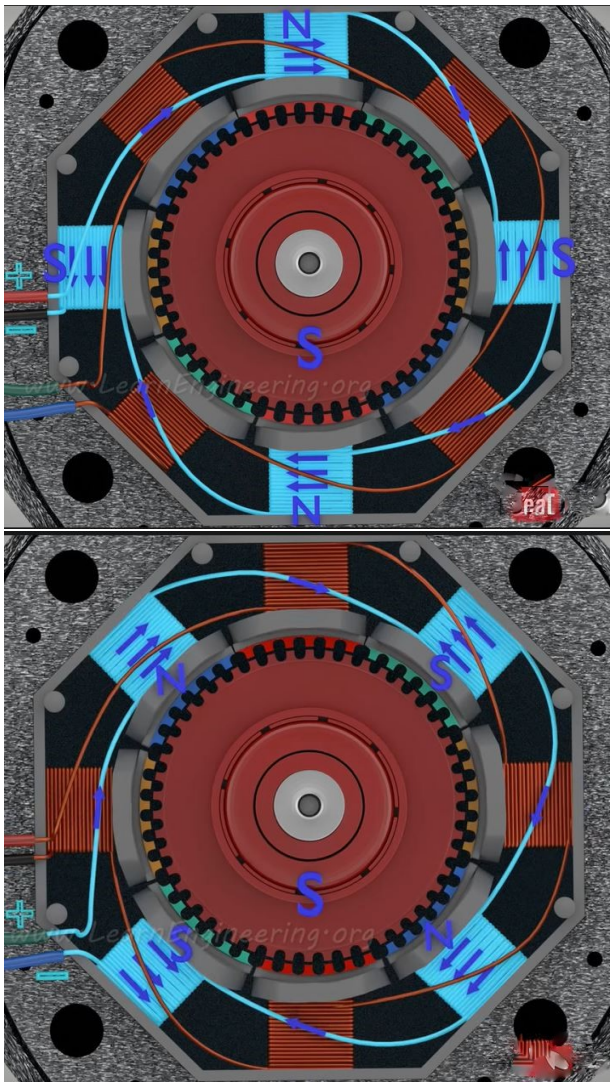

Figure 5: Working hybrid stepper motor

The rotor is a magnet with a north and south pole. The pictures above are showing the site of the south pole. If we supply a current to the red and yellow coils so they form a North and south pole.The teeth of the rotor will align with the (red) north pole and move away form the teeth of the (yellow) south pole. The motor will rotate exact 1.8°. Blue and green teeth are half aligned with the rotorteeth. If the current now flows to the blue and green coils, forming a new North and south pole. The rotor will turn an other angle. Than the same when doing this over with changed polarities.
With 50 teeth, one angular pitch (360°/50) is 7.2°. One step on the other hand is 1/4 of an angular pitch. This is why we have a step angle of 1.8° or 200 steps per rotation. [15] [5]

3.2 Electrical characteristics

The motor works on a voltage of 2.5 V and a current of 4.5 A. To get the full power out of the motor we need to supply the motor with 4.5 Amps. The voltage is just calculated with the resistance of the motor and is less important. We can power the stepper with a voltage of 24 volts and reuse the previously used power supply of the wind tunnel.
The motor has a step angle of 1.8°. So this are 200 steps per rotation. It is possible to get an higher amount of steps by using a driver that support microsteps.
The holding torque is 2200 mNm, this indicates that when the motor is energized 2.2 Nm are needed to move the rotor a full step. If the windings are not energized, only 100 mNm is needed. [1] [16]

The electrical characteristics can also be seen in the table below.

Table 1: Electric characteristics RS 440-470

RS stock no.	440-470
Rated voltage (V)	2.5
Rated current (I)	4.5
Resistance (Ω)	0.56
Inductance (mH)	2.8
Detent torque (mHm)	100
Holding torque (mNm)	2200
Step angle accuracy (%)	5
Step angle	1.8
Insulation class	B

4 Inter-Integrated Circuit

4.1 What is I²C

The Inter-Integrated Circuit (I²C) is an communication bus to transfer data between devices. It follows a protocol that allows multiple peripherals to communicate with one or more controllers. It is intended for short distance communication and only requires two connections to exchange data. I2C is synchronous so master(s) and slave(s) share one clock signal. [3] [6]

4.2 Why I²C

The I²C protocol has a lot of advantages and is perfect for projects with a lot of peripherals. This protocol only uses two connections and supports up to 128 slaves and theoretically infinite masters. It is easy to expand in the future by adding new components, what makes it very flexible. To really see why this protocol is the best for this project we can compare I²C to some other common used options. In the following part you can see why I²C is better than UART asynchronous serial communication and SPI protocol. [8]

4.2.1 UART

UART stands for Universal Asynchronous Receiver/Transmitter and is for asynchronous serial communication. Therefore there is no clock data and all devices need to agree on a data rate and must have clocks with a close to similar rate. Asynchronous serial ports require hardware overhead and implementing it precise in software is difficult. The data requires at least one start and stop bit for every transmitted data what which eats into the data rate. The biggest disadvantage is that serial ports are suited to communications between only two devices. It is possible to connect multiple devices, but this usually requires external hardware. Bus contention where two devices attempt to use the same line at the same time is always a problem and must be dealt with carefully to prevent damage. Finally the data rate is an issue. UART devices only support a specific range of fixed baud rates, mostly the highest of them around 230400 bits per second. [8]

Figure 6: UART communication

4.2.2 SPI

With Serial Peripheral Interface or SPI protocol three common connections are needed and one extra chip select I/O pin on the controller for every peripheral. For one controller and one peripheral you only need four connections, but for every extra peripheral the quantity of connections grows bigger by one. This makes the protocol not so flexible for future extensions. SPI allows just one controller on the bus. The protocol is good for high data rate full-duplex connections. This means simultaneously receiving and sending data over the bus. It supports clock rates up to 10Mhz and has fast speeds. Implementing SPI in software is easier than with UART. [8]

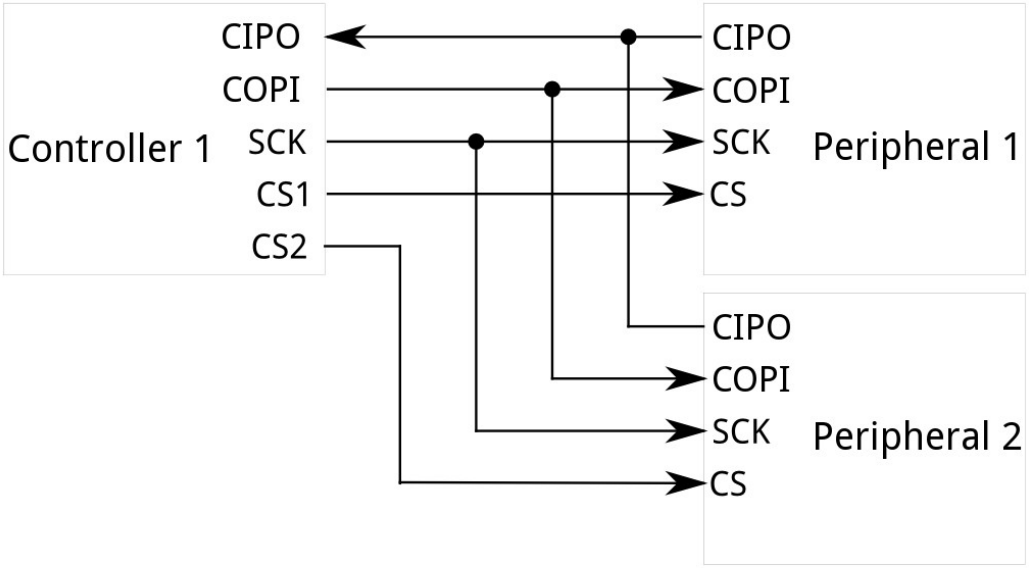

Figure 7: SPI communication

4.3 How does it work

The Inter-Integrated Circuit protocol combines the best features of the SPI and UART protocols. Like mentioned before, I2C works on two single wires. Serial Data (SDA) and Serial clock (SCL). Over the SDA connection the data gets transferred between the controllers and slaves. The SCL connection takes care of the clock signal to synchronise the data transfer between the devices. [6]

4.3.1 data transfer

Data is transferred in messages that can be split up into 8 bit sequences of data, sent over the SDA connection. Like shown in picture 8 below. Only a master can start a conversation. The data stream starts with a start signal bit, to start the communication between two devices, flowed by the address of the slave. After the slave address there follows, like after every 8 bit sequence, an acknowledge (ACK) bit. The sending device will make the SDA line high and if the the data is well received by the receiving device, it will make the signal back low. If the signal stays high, than the sender knows that message was not properly received. This is called "not acknowledge". The second sequence contains the 8 bit internal register address of the slave. followed by an other ACK bit. At least we

have the data split in packages of 8 bits plus an ACK bit. The data is always send with the most significant bit first. If all the data is sent the communication is stopped by a stop signal bit. Now a new connection between two devices can be made over the bus. A special thing about the start and stop bits is that they gets sent under a condition. The start condition is that the SDA signal gets low when the SCL signal stays high. Like shown at the bottom of picture 8, this can not happen when two devices are already communicating. With the stop bit the SDA signal needs to get high when the SCL signal is also high. Now both lines are high and an other master can start a connection by making the SDA line low again.

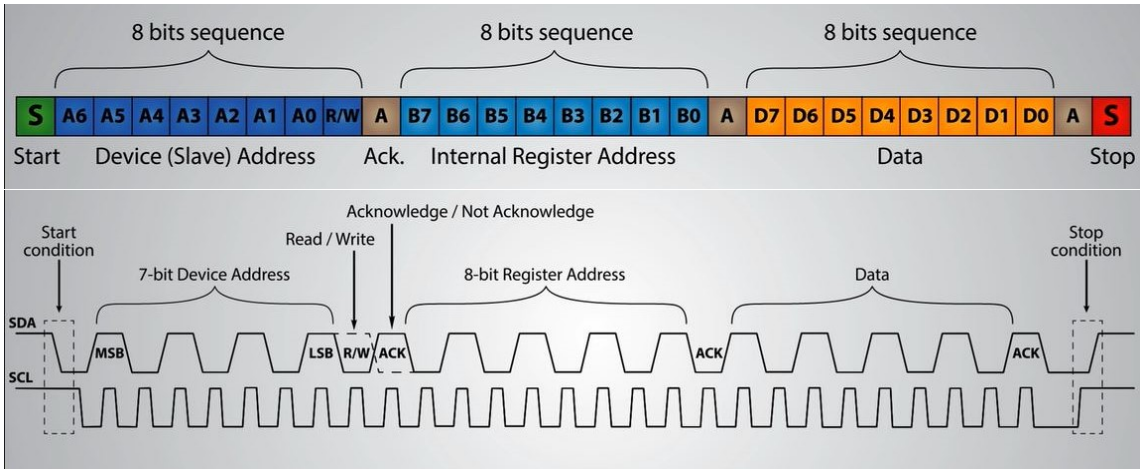

Figure 8: I2C data structure

There is also a phenomenon called "Clock Stretching". Here the controller takes longer than a normal clock pulls to pull the ACK bit high. This is when the data rate of the controller exceed the ability of the peripherals to provide that data. When the data is not ready yet or the precious operation is not finished yet. All clocking is driven by a controller and the peripherals only take/put data from/on the bus. Unfortunately, this slows down communication over i2c. This can make it more difficult to control the stepper motors. on the other hand, this will not be a problem if there are only a few devices on the bus. [8] [3] [6]

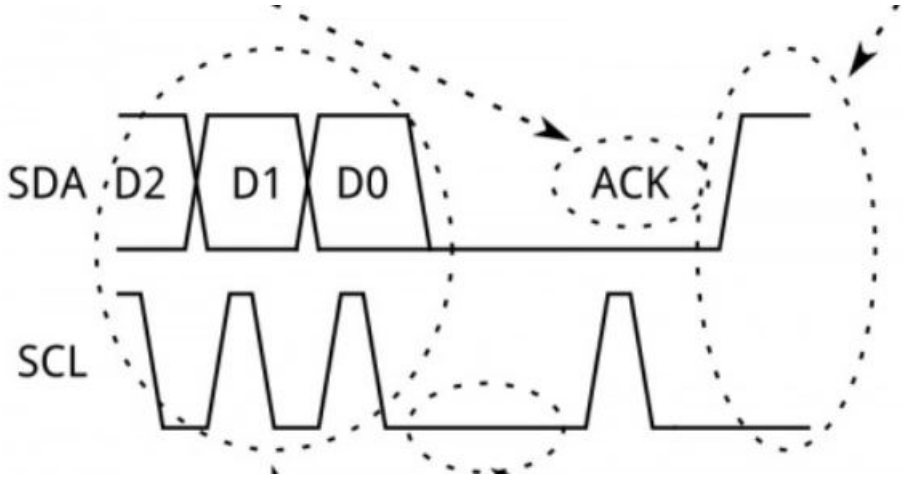

Figure 9: Clock stretching

4.3.2 addressing

I2C doesn't have a slave select lines like SPI. To know which slave needs to respond, we use addressing. Every slave has a unique address and compares it to the address frame that comes directly after the start bit. If the address matches, the slave sends a low voltage ACK bit back to the master and if not the SDA line remains high. The last bit of this address is reserved to indicate if the master Writes to the slave (logic 0) or read from the slave (logic 1). So the right slave knows what to do, read and use the data form the master or send data to the master. [8] [3] [6]

4.3.3 wiring

The wiring is quite easy because it only contains two wires. All the devices should be connected to the SDA - and SCL wire in parallel. Both wires also need to be connected to a power supply. The I2C circuit can take 3mA of current. If we use the output voltage of the Raspberry Pi 4 that goes to 3.3V, we need to put it in series with a resistor of minimum 1.1KΩ. The selection of the resistors depends on the devices on the bus. There is a rule of thumb that says to start with 4.7KΩ and adjust down is necessary. For long distances or lots of devices, a smaller resistor is recommended. [3]

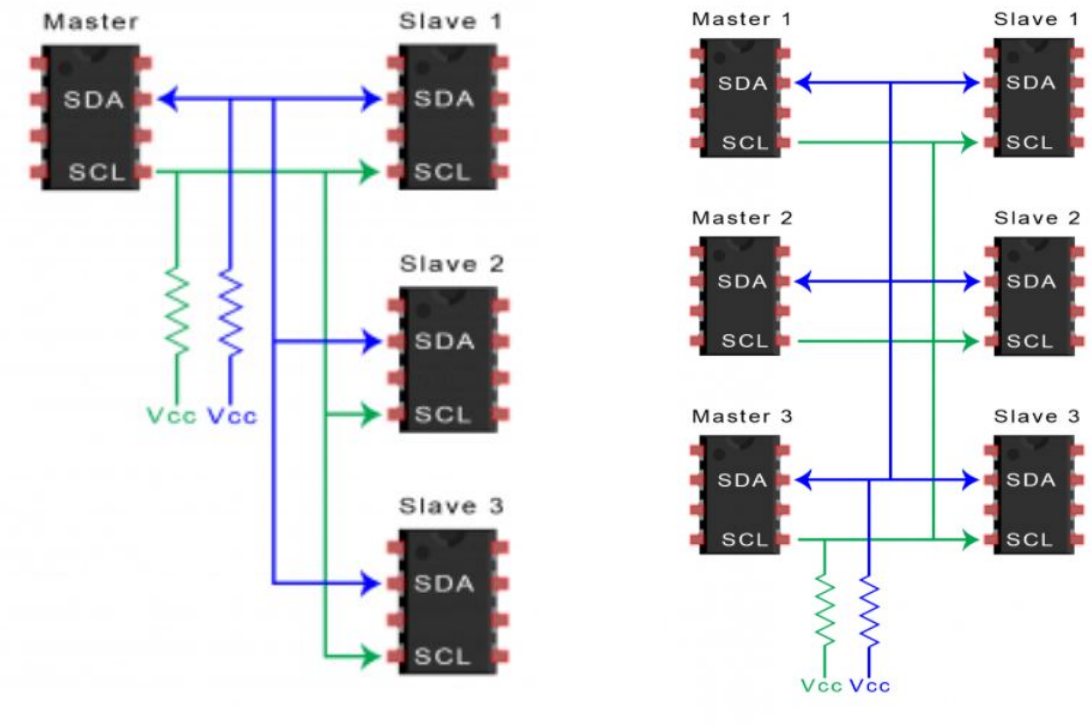

Figure 10: Wiring master(s) and slaves

5 Project requirements

5.1 Components

Controller: Raspberry Pi 4 [4] [11] [13]

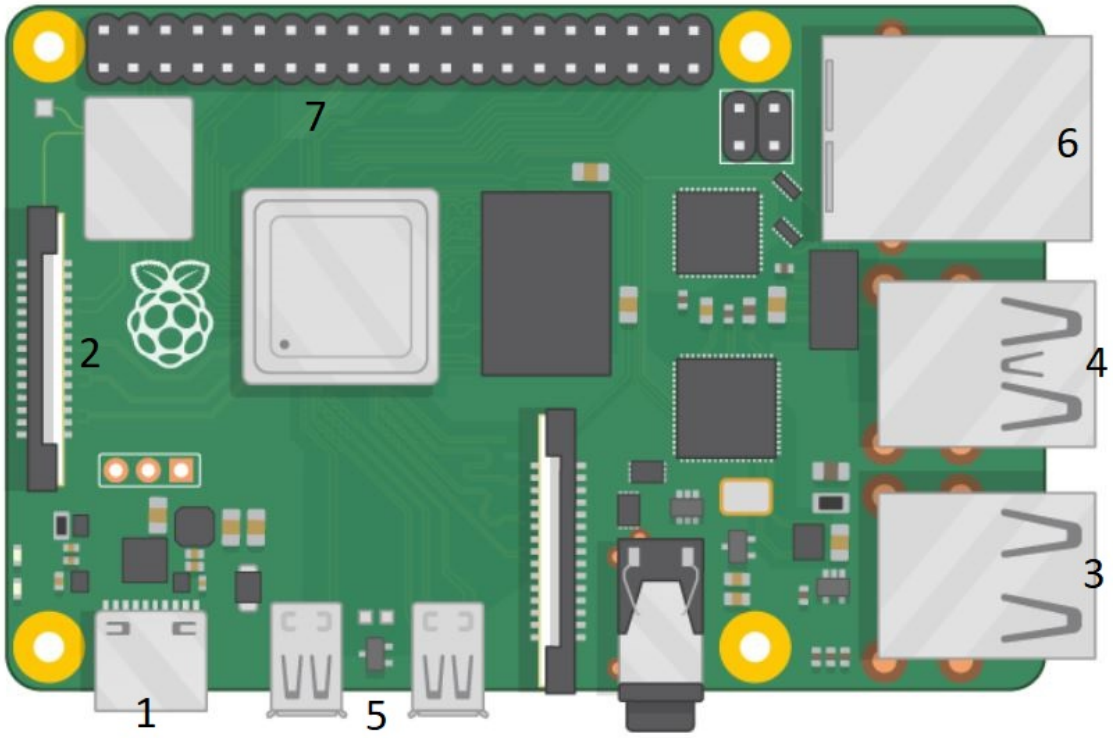

Figure 11: Raspberry Pi 4

The Raspberry Pi 4 Model B is at this moment the newest, fastest and easiest to use. This makes it the best option considering the performance throughout future. A RAM of 2GB will be enough for this project. Following are the connections of the Pi 4 useful for this project.

1. USB-C port for power
2. microSD card slot for the memory
3. 2x USB 2.0 ports
4. 2x USB 3.0 ports
5. 2x micro HDMI ports
6. Gigabit ethernet port
7. I/O pins

In the next picture you can see the I/O pins of the Raspberry Pi 4.

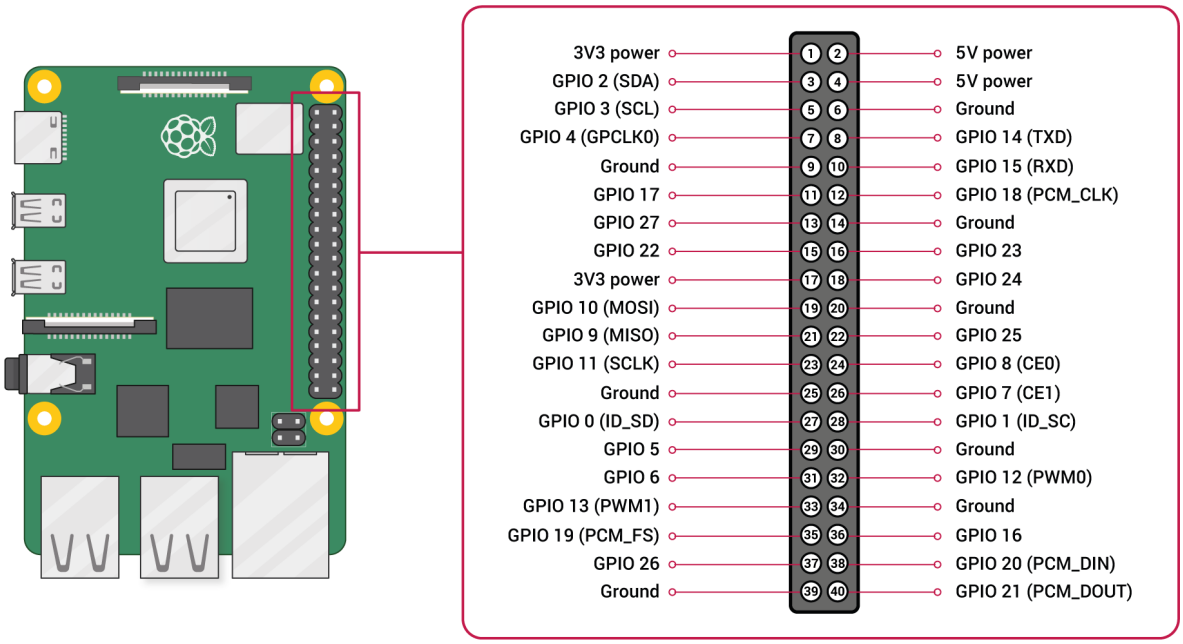

Figure 12: description of the In/Out-pins

In the I/O pins we need pins 2 (SDA) and 3 (SCL) for the I2C connection. For the I2C protocol we also need a voltage and we can use the 3.3V output of the Raspberry Pi.

Other components required for the Raspberry Pi are:
Power plug, mouse and keyboard, micorSD card (minimum 8GB), monitor, Ethernet cable

Theoretically, the Pi can send pulses to the motor driver faster than most any drive could accept them. But like mention before, the I2C protocol slows this pulses down. This will not be a problem is the raspberry pi don't get to many tasks.

Driver: Grove - I2C Motor Driver V1.3 [2]

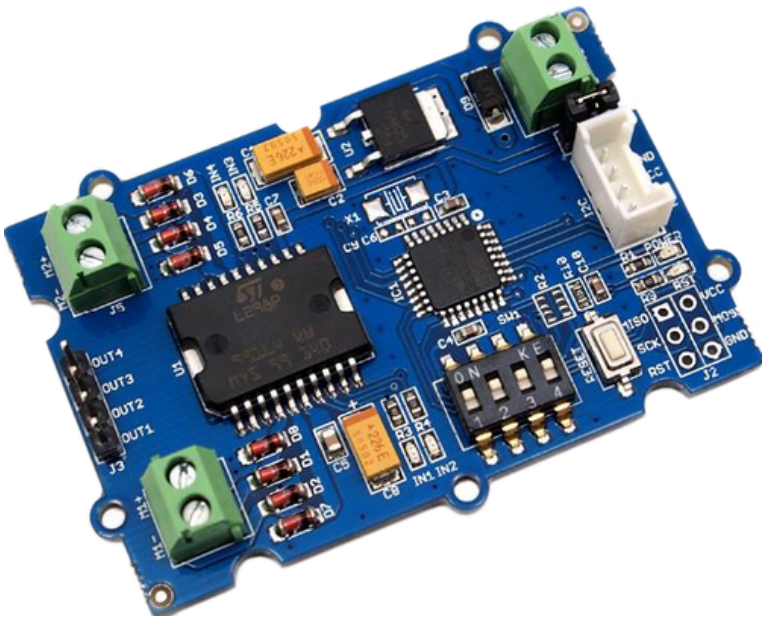

Figure 13: Grove I²C Motor driver

The Grove - I2C Motor Driver V1.3 can directly control a stepper motor or up to two DC motors. It can handle up to 2A per channel and supports the I2C communication. As a power supply it needs 6V to 15V. IT has an 5V voltage regulator inside that can provide a fixed output voltage for the I2C bus.

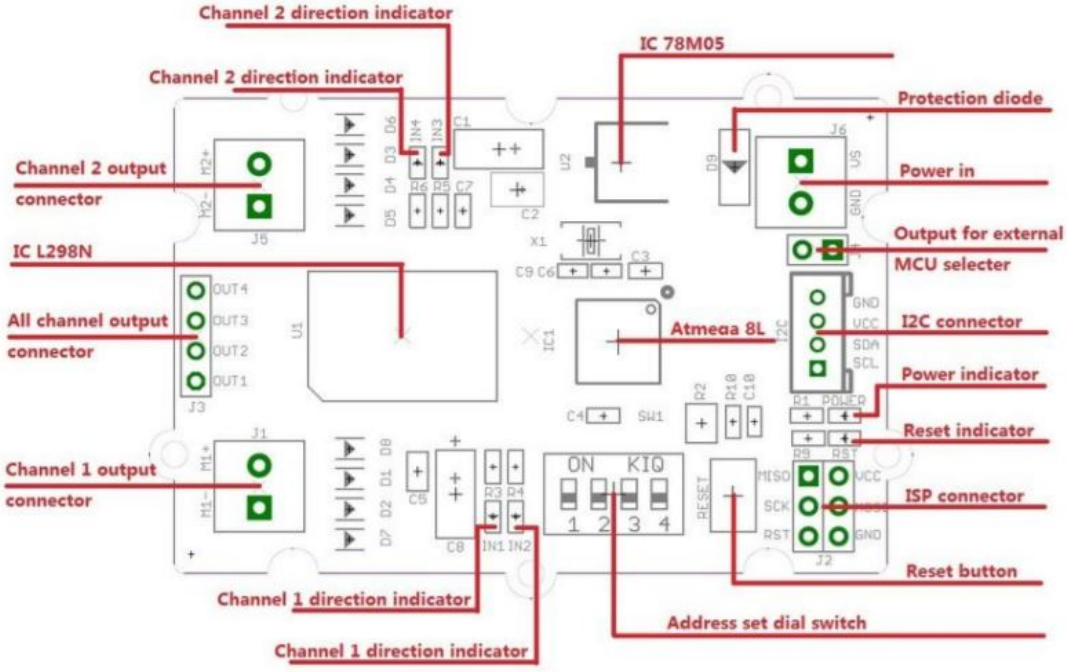

Figure 14: Grove I²C Motor driver connections

Motors: RS 440-470 stepper motor
The motors are going to be reused from the wind tunnel and therefore are already in possession at the start of this project. These are discussed in chapter 3. The motor coils can be connected in series or parallel.

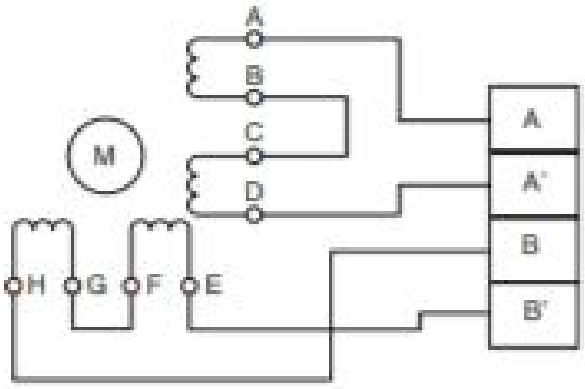

Figure 15: 8 wire connection in series

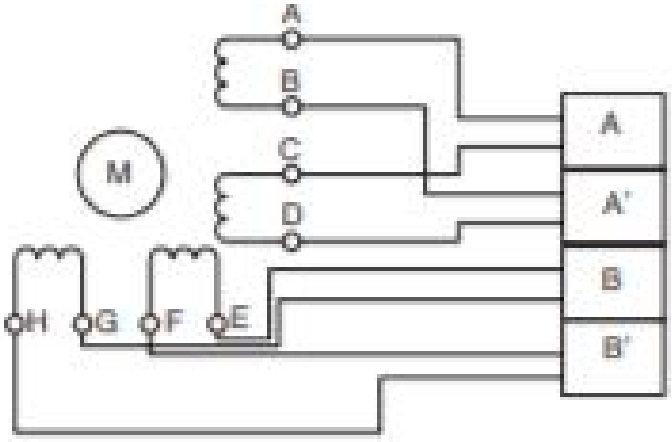

Figure 16: 8 wire connection in parallel

5.2 schematics

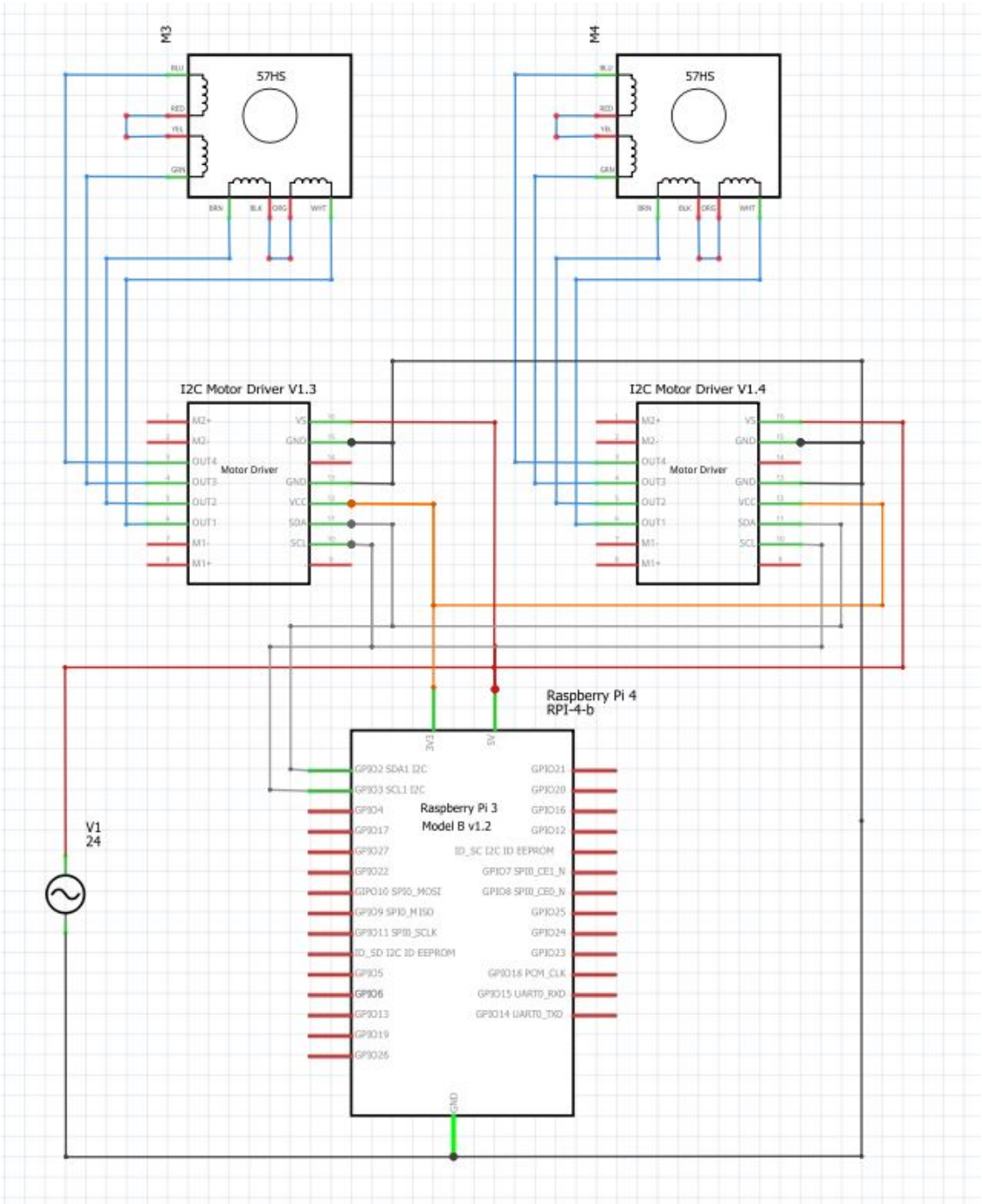

Figure 17: Wiring schematic

6 Raspberry Pi setup

6.1 necessary hardware

To setup the Raspberry Pi a few items are required. First of all to power the Pi, a charger with usb-c is needed. This one should give a -+5V output with a current of at least 3A for the Pi 4. A USB keyboard and mouse are needed for the first setup. Later if wanted, these can be replaced with a Bluetooth mouse and keyboard. To view the OS desktop environment, a screen is necessary but not if you're not planning on using the Pi as a computer. For monitors the Raspberry Pi 4 has two micro HDMI ports. For a screen with HDMI output, a micro HDMI to HDMI cable can be used. The most important item is a micro SD card with at least 8GB of memory. The Pi needs an SD card for the operating system and to store all its files.

6.2 Software

To get the Raspberry Pi OS software on your memory card, you need a computer with a SD card port and download the software online. The easiest way to do it is to install the Raspberry Pi Imager online via the link: "https://www.raspberrypi.org/downloads". Install the software for the right operating system on your PC and install it by following the instructions. [12]

Figure 18: Pi OS download start screen

When you launch the software you get to see a screen like in the picture above. You need to choose the right operating system and the right SD card to wright the data to. [10]

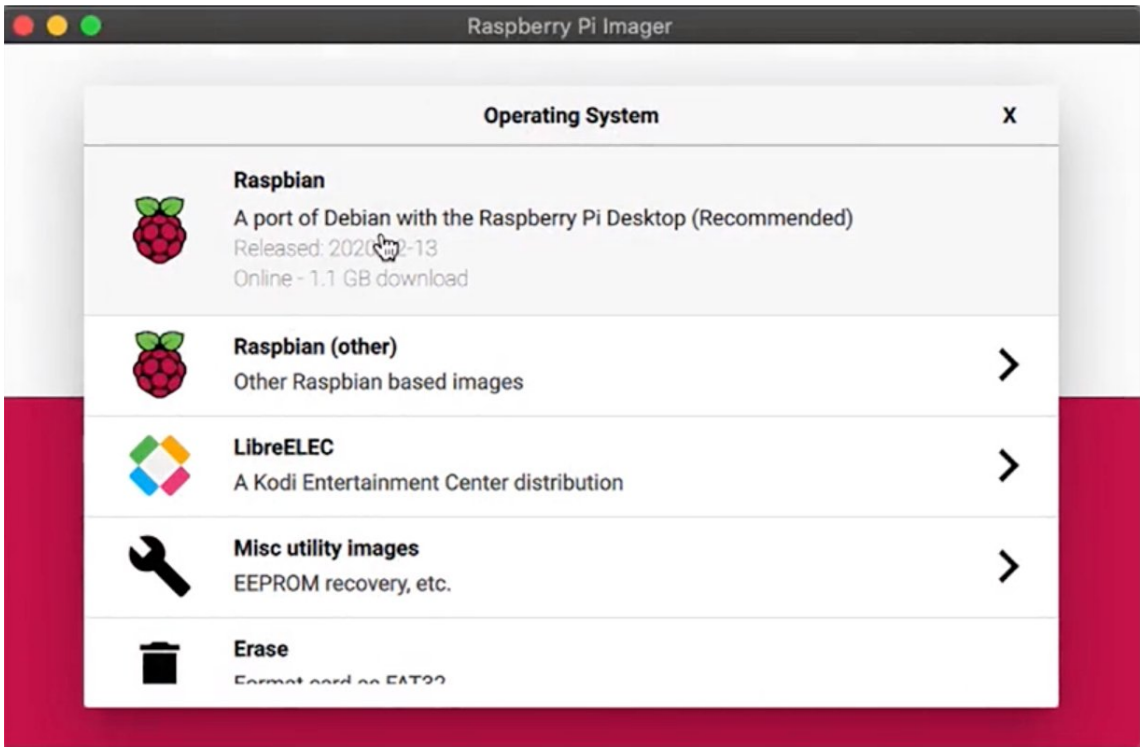

Figure 19: Pi OS download step 1

For the operating system, chose the recommended one. This one is the most used and works really good. [10]

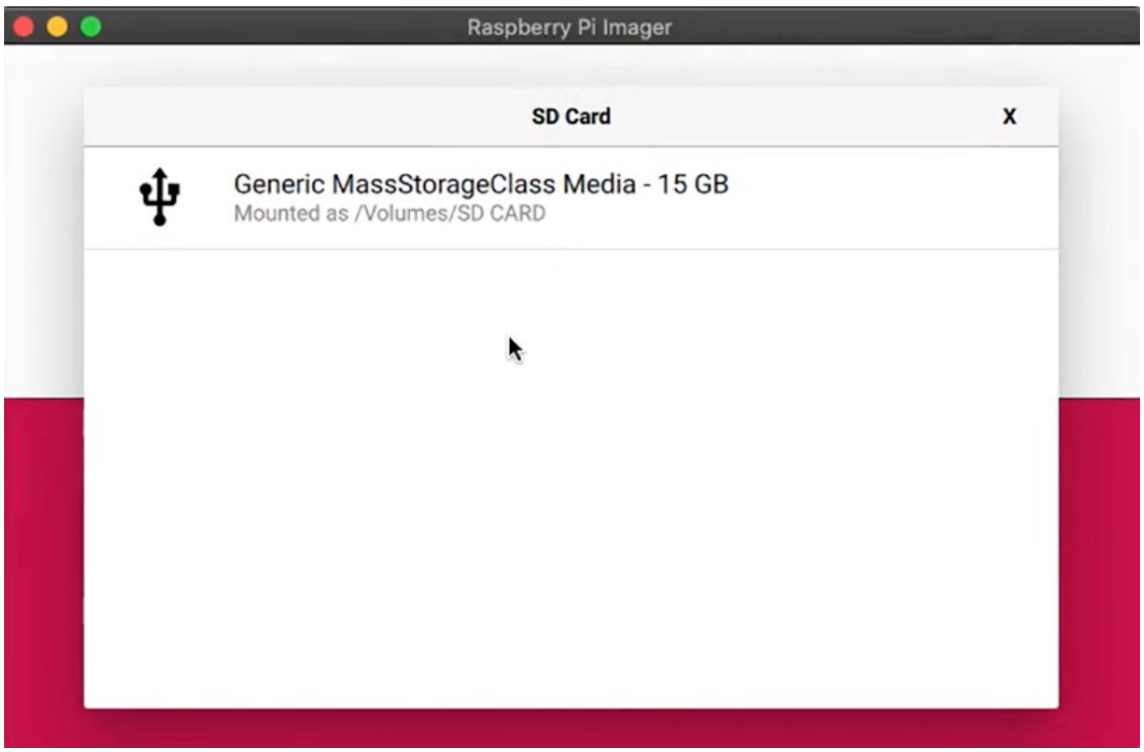

Figure 20: Pi OS download step 2

Chose the right port of the computer where the SD card is plugged in and everything is ready to download. [10]

Figure 21: Pi OS download step 3

The only thing left to do is click on the "write" button and wait till the download is finished. [10]

6.3 Power up

After inserting the SD card in the Raspberry Pi and connecting the monitor, keyboard, mouse and Ethernet cable we can plug him in the power supply. After a couple seconds the Pi will start up and ask which OS you want to install. Select the first one "Raspbian full". It is the most popular version and recommended by Raspberry Pi. At the bottom of your screen, you can change the language and the keyboard settings. Now click on install and wait till you get a confirming message. [7] [12]

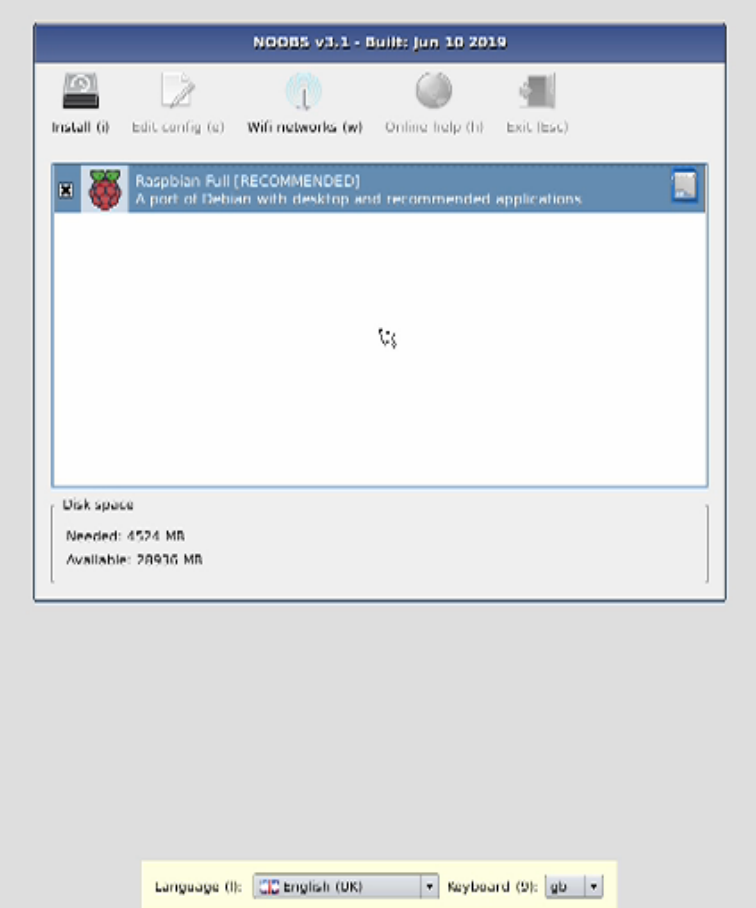

Figure 22: Install OS on Raspberry Pi

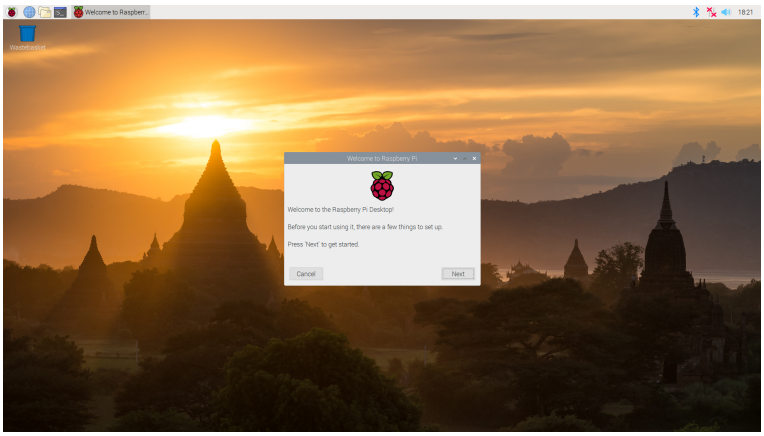

Figure 23: Raspberry Pi desktop

When the installation in ready, click next an select your time zone and language. Create a login password and you are almost ready to go. You can also connect to a WiFi network as wanted.

Figure 24: Select country, language and timezone.

Figure 25: Create password

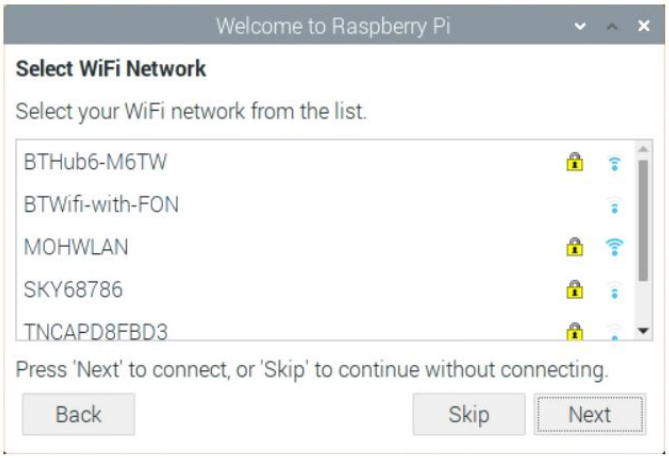

Figure 26: Select WiFi network

At least you get a update screen. To update to the latest version, if necessary, you can

click "Next" or click "Skip" to skip this part.

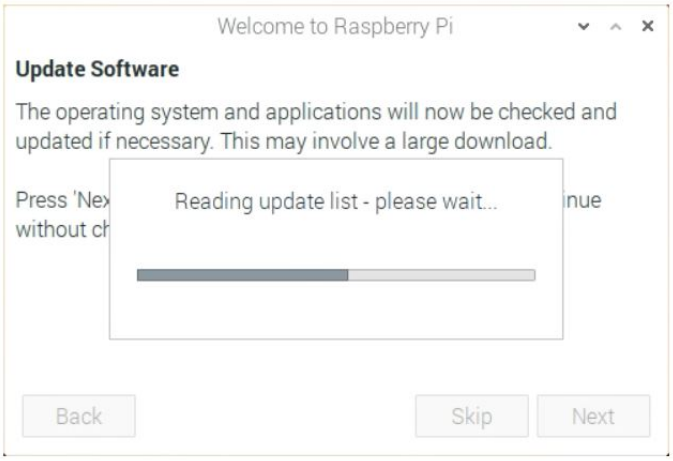

Figure 27: Look for updates

When connected to the internet you can look for updates in the terminal. Open the terminal and write the code: "sudo apt-get update". With the code "sudo apt-get upgrade -y" you update to the latest version. This is if you aren't using the newest one yet.

7 Programming

Python is the programming language in which the project will be programmed. Python is Free and Open-Source. It has a large community and is easy to use and to learn. The standard library of Python is very big, you can find almost all the functions needed for your task.

7.1 Configure Your Pi

For this project we want to use Python 3. To check the version the Raspberry Pi is using, type this code in the terminal.

```
python –version
```

If you see a version lower than 3, like Python 2.7.13 for example, enter the next commands. this will change the version of Python to Python 3.

```
nano  /.bashrc

alias python='/usr/bin/python3'

source  /.bashrc
```

At least you can chack the version again. It chould say Python 3.x.x .

```
python –version
```

Install pip

Pip is a package management system used to install and manage software packages. We are using the full desktop version of Raspbian so it is already installed.

7.2 setup I2C connection

Enabling

The I2C protocol is not enabled by default, we need to adjust the settings to turn it on. Go to the **Pi start menu** → **Preferences** → **Raspberry Pi Configuration**

When the setings tab pops up we want to go to the **Interface** tab. Than select **enable** for I2C

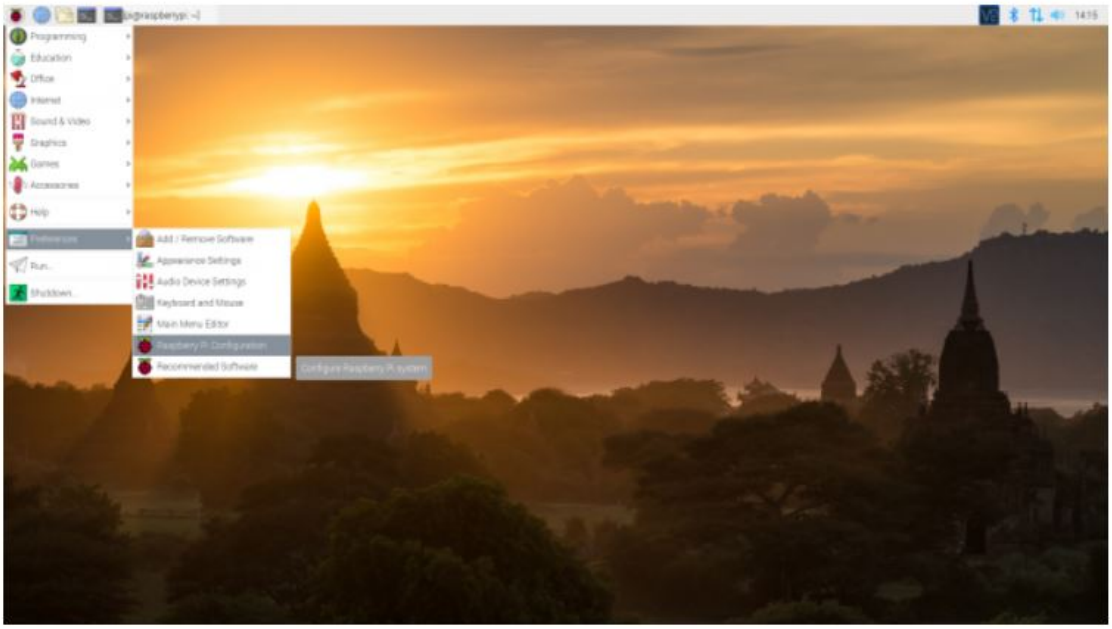

Figure 29: Enabling I2C

To make sure the changes worked out, the Pi should be restarted. **Pi Start Menu** → **Preferences** → **Shutdown** ⟶ *pick* **Restart**.

Utilities

To help getting the I2C interface working we can use a set of command-line utility programs.

```
sudo apt-get install -y i2c-tools
```

```
i2cdetect -y 1
```

We put the -y flag so we don't have to wait for confirmation. The 1 indicates that we are scanning for devices on bus 1 of the I2C. The output will look like the picture below. We can deduce that there is a peripheral whit address 0x60 on the bus.

```
pi@raspberrypi:~/$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: 60 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

COPY CODE

Figure 31: Example, output giving peripheral address

7.3 code: I2C connection

The first important thing is that in Python hexadecimal format is used for I2C buses. There are several ways to form an 8-bit number. You could use for example binary to show each of the 8 bits individually 0101 0101. In decimal the same number would be shown as 125 and in hexadecimal the number would be shown as 55. To wright down hexadecimal code in Python, you put an "0x" before the number. For example 0x55.

For the I2C connection we need to import the SMBus library.

```
from smbus import SMBus
```

In the table below is a summary of the functions included in this library. [9]

Table 2: SMBus: summary of available functions

Function	Description	Parameters	Return value
SMBus Access			
write_quick(addr)	Quick transaction.	int addr	long
read_byte(addr)	Read Byte transaction.	int addr	long
write_byte(addr,val)	Write Byte transaction.	int addr,char val	long
read_byte_data(addr,cmd)	Read Byte Data transaction.	int addr,char cmd	long
write_byte_data(addr,cmd,val)	Write Byte Data transaction.	int addr,char cmd,char val	long
read_word_data(addr,cmd)	Read Word Data transaction.	int addr,char cmd	long
write_word_data(addr,cmd,val)	Write Word Data transaction.	int addr,char cmd,int val	long
process_call(addr,cmd,val)	Process Call transaction.	int addr,char cmd,int val	long
read_block_data(addr,cmd)	Read Block Data transaction.	int addr,char cmd	long[]
write_block_data(addr,cmd,vals)	Write Block Data transaction.	int addr,char cmd,long[]	None
block_process_call(addr,cmd,vals)	Block Process Call transaction.	int addr,char cmd,long[]	long[]
I²C Access			
read_i2c_block_data(addr,cmd)	Block Read transaction.	int addr,char cmd	long[]
write_i2c_block_data(addr,cmd,vals)	Block Write transaction.	int addr,char cmd,long[]	None

We also need to pause the program by using the time.sleep function out of the next library.

```
import time
```

Defining the Registers

Inside the main() function we add a list with the register addresses. These go from 0x00

till 0xXX, depending on how many registers are used.

Creating an SMBus Object
This SMBus object will represent the physical I2C bus of the Pi. All commands to send and receive data though the bus go through the SMBus object. We will call this object: "i2cbus":

```
i2cbus = SMBus(1)bus
```

Device Address
Each device on the I^2C bus has a unique 7-bit address. Create a variable called "i2caddress" with a value of 0x00. This is the default address of the first peripheral.

```
i2caddress = 0x00
```

Reading and Writing to the Ports For reading and writing to the ports we will create a infinite loop.

```
while (True):
```

To read data from the bus we use the read_byte_data function. We make this equal to the desired port.

```
"portX" = i2cbus.read_byte_data(address, register)
```

To write a value to a port we need the write_byte_data function.

```
i2cbus.write_byte_data(address, register, port)
```

conclusions

The aim of this project was to renew the wind tunnel in the TR5 building of the ESEIAAT campus of UPC Terrassa. To make the project as universal as possible and leave options open for the future, certain choices were made in the selection of new hardware, software, the protocol used and the coding. There are huge data bases and plenty of tutorials online for Raspberry Pi what makes it the best controller to easily change/expend thing in the future. Python is commonly used what makes it the perfect coding language. I2C was chosen as the communication protocol to limit the number of wires and to make it easy to add new components to the project. One disadvantage was that the I2C protocol could be to slow for controlling stepper motors. Nevertheless can this problem be avoided if there are not to many peripherals on the bus and/or if the Raspberry Pi doesn't have to many tasks.

Due to various reasons, the practical side of this project was not completed and this thesis became more of a theoretical thesis.

References

[1] 0900766b81579a99.pdf. https://docs.rs-online.com/68bd/0900766b81579a99.pdf. (Accessed on 06/19/2021).

[2] 105020001_01.pdf. https://cdn-reichelt.de/documents/datenblatt/A300/105020001_01.pdf. (Accessed on 06/20/2021).

[3] Basics of the i2c communication protocol. https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/. (Accessed on 06/19/2021).

[4] Gpio - raspberry pi documentation. https://www.raspberrypi.org/documentation/usage/gpio/. (Accessed on 06/19/2021).

[5] How does a stepper motor work ? - youtube. https://www.youtube.com/watch?v=eyqwLiowZiU. (Accessed on 06/19/2021).

[6] How i2c communication works and how to use it with arduino - youtube. https://www.youtube.com/watch?v=6IAkYpmA1DQ. (Accessed on 06/19/2021).

[7] How to set up raspberry pi 4 — the magpi magazine. https://magpi.raspberrypi.org/articles/set-up-raspberry-pi-4. (Accessed on 06/22/2021).

[8] I2c - learn.sparkfun.com. https://learn.sparkfun.com/tutorials/i2c. (Accessed on 06/19/2021).

[9] I2c part 4 programming i²c with python with raspbian linux on the raspberry pi. https://www.abelectronics.co.uk/kb/article/1094/i2c-part-4---programming-i-c-with-python. (Accessed on 06/22/2021).

[10] Introducing raspberry pi imager, our new imaging utility - raspberry pi. https://www.raspberrypi.org/blog/raspberry-pi-imager-imaging-utility/. (Accessed on 06/22/2021).

[11] rpi_data_2711_1p0_preliminary.pdf. https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf. (Accessed on 06/19/2021).

[12] Setting up your raspberry pi - introduction — raspberry pi projects. https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up. (Accessed on 06/22/2021).

[13] Setting up your raspberry pi - what you will need — raspberry pi projects. https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up/1#:~:text=Raspberry%20Pi%204%20has%20two,Pi%204%20to%20a%20screen. (Accessed on 06/19/2021).

[14] Stepper motor - wikipedia. https://en.wikipedia.org/wiki/Stepper_motor. (Accessed on 06/19/2021).

[15] Stepper motors: Types, uses and working principle — article — mps. https://www.monolithicpower.com/en/stepper-motors-basics-types-uses. (Accessed on 06/19/2021).

[16] untitled. https://docs.rs-online.com/c047/0900766b81579a98.pdf. (Accessed on 06/19/2021).

Appendix: circuits schematics's

Electronics wiring schematic :