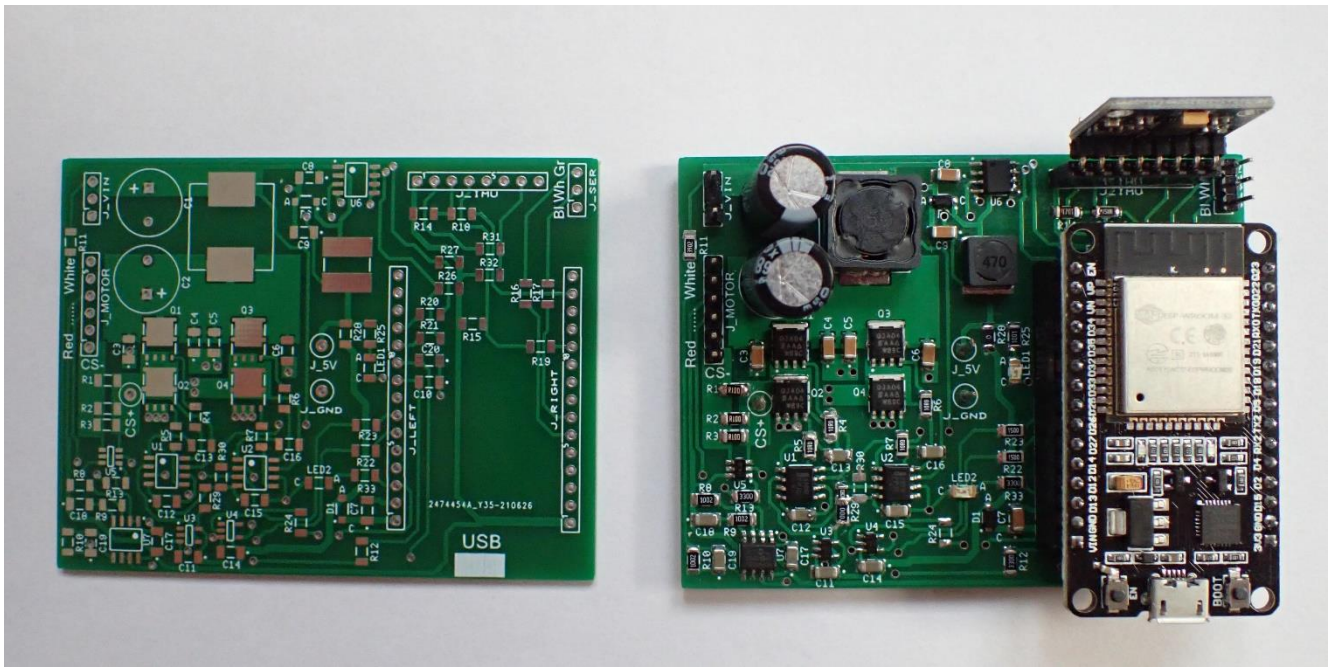
HAN_UNIVERSITY OF APPLIED SCIENCES

Minor Mobile Robotics

Reader Robotic electronics



Stefan van Sterkenburg

August 2025

Version 4

Contents

Introduction

General

In the course electronics for robotics you learn how to design, make and test a low level controller that can be used in a robotic applications. As a case, you have to develop a motor controller that can be controlled via serial communication and via a TCB/UIP socket connection with a high level controller.

The motor controller exists of hardware (electronics) and software running on an embedded system. In this course both aspects will be discussed.

The learning principle of this course is “learning by doing”. This means, that you will learn how to design the hardware and embedded software by making and testing it. The following topics are dealt with:

1. The design of the electronic circuits, among which sensor circuits, a full H-bridge to control the motor, a switch mode power supply and some basic interface electronics.
2. The selection of electronic components (matching requirements and datasheets)
3. The designing a printed circuit board using a professional PCB-CAD tool.
4. The testing of the electronics circuits and embedded software
5. The interfacing the electronic circuits with an MCU (we use an ESP32 and micro python)
6. The implementation of communication interfaces of the motor controller and a high level control unit.
7. The writing of embedded software to control the electronics

The whole process of the designing, making and testing of the motor controller is carried out via a set of 26 questions and assignments. The table below shows the contents of the lectures and practical workshops that are organized to support you in making these questions and assignments.

Lectures

Week	Topics discussed in lessons	Questions to be made
1	Basics of EasyEDA Basics of Network theory DC-motor operation	1, 2
2	Power electronics Analogue opamp circuits (1)	3, 4, start with 11
3	Analogue opamp circuits (2)	5, 6, 7, 8, 9, 10, 11
4	Micropython, Thonny IDE, Linear regression with Numpy	12 up to 16
5	I2C, Serial communication	17 up to 22
6	TCP-communication	23, 24
7	Upon request	25, 26

Assessment

The course is assessed by:

1. The elaboration of the 26 questions of this reader. For each assignment you can get a certain number of points with a total of 122. The final grade is the total number of points earned divided by 12.2.
   The questions of the reader must be worked out in couples of 2 students. The answers on the question must be worked out in the Answer_document.docx (Brighspace)
   The Osiris code associated with this is TOETS-05.
2. A written exam which normally takes place in week 9 the term. The Osiris code of the exam is TOETS-06.

Deliverables

The following must be turned in at latest on Friday of week 9.

- The completed answer document containing the answers to the questions in this reader.

- A folder that holds the EasyEDA project (both the schematics as the PCB-design).

- Folders that hold the python script files of questions 13, 14, 15, 16, 18, 21, 22, 23, 24, 25 and 26. Each folder must have all scripts that are necessary to run the application.

All documents must be stored in the folder 'Answers_MR_ELEC' that you can download from Brightspace. This folder holds an empty word answer document and subfolders for the EasyEDA project and python scripts. Add only one version (the latest version) of each deliverable otherwise the wrong version might be assessed. At the end, you have to zip the folder with deliverables and turn the zip-file in via Handin.

Recommended literature

[1] Basics electronics.pdf (Brightspace)

[2] Power_Circuits.pdf (Brightspace)

[3] Analogue opamp circuits (Brightspace)

[4] Python Programming 2024 (Brightspace)

1 Motor controller specifications

In almost each robotic system you find one or more electric motors. In this course we show how to design the (electronic) hardware and embedded software of a motor controller. The motor controller is a low level electronic device that controls a motor and is fed by a power supply. The motor controller itself is often controlled by a high level controller, such as a computer or PLC (see figure 1.1).

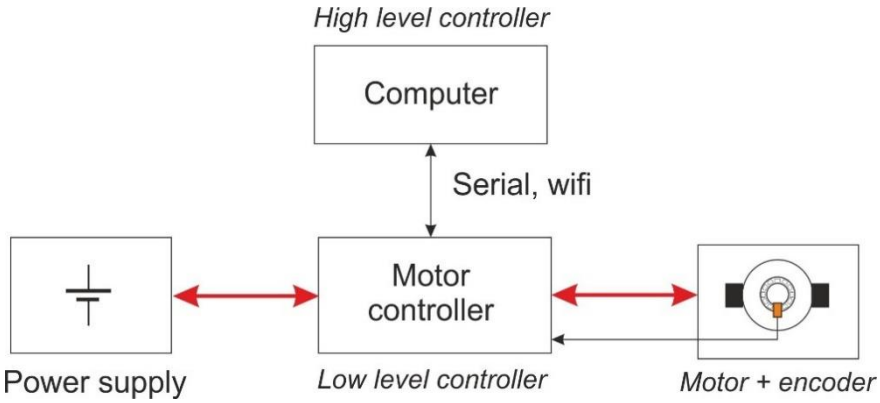


figure 1.1: Block scheme that shows the control of a DC-motor by a high level controller via a motor controller.

The following functional requirements apply to the motor controller that is discussed in this course:

1. The motor controller is used for a 12V brushed DC-motor.
2. The motor controller has an input for a quadrature encoder.
3. The motor controller enables 4 quadrant operation (motor and generator operation in 2 directions).
4. The motor controller has a speed control combined with a torque limit control.
5. The motor controller has a current and voltage sensor.
6. The motor controller can be interfaced via wifi or serial communication.
7. The motor controller has an input for a 6-DOF gyroscope/accelerometer that is used for tilt protection.
8. The motor controller has some basic user-interface related hardware (3 buttons, an RGG-led and a buzzer).

The following technical requirements apply:

1. The supply voltage range of the motor controller is from 9V-15V.
2. The maximum supply and motor current is 3 [A].
3. The motor voltage is adjustable from 0V to at least 95% of the supply voltage.
4. The inaccuracy of the voltage and current sensor is at most 2% of full scale.
5. The motor controller has an operating temperature range from -20°C to 85°C.
6. The motor controller has a wifi and serial port communication interface.

The following constraints apply:

1. An ESP32 MCU board is used.
2. If possible, automotive qualified components are used.
3. The cost price of all electronics may not exceed €15 (this does not include the ESP32 board).
4. The maximum size of the PCB is 80mm x 120mm.

Question 1 (6 pnts)

In this course, we will test the motor controller with the following motor and encoder: RB-Dfr-444 (see: 12V DC Motor 251 RPM met encoder - RobotShop).

Find information on internet information about this motor and encoder. If necessary, find also on internet some basic theory about DC-motors and quadrature encoders in general (page 13 and 14 of [2] explains very briefly the working of a DC-motor). Then, answer the questions below:

a. Give the equivalent network scheme of a brushed DC-motor.

b. Give the equation that expresses the back EMF in [V] as a function of the angular speed of the motor in [rad/s] and the equation that expresses the motor torque in [Nm] as a function of the motor current in [A].

c. Determine the armature resistance of the motor from the data given in the datasheet.

Hint: Use the stall current. The stall current is defined as the current when the motor is connected to the rated voltage and is not running. Because the motor is not running, there is no back EMF and the stall current is only limited by the armature resistance.

d. Calculate the motor constant K_m of the motor.

Hint: Use the equivalent network schema of question a to calculate the back EMF at the rated rpm and rated torque. Then use the equation found in question b to calculate the motor constant.

e. Determine the efficiency of the transmission.

Hint: Use the relationship:

$T_{shaft} = \eta \cdot i \cdot T_m - T_{sf}$

Where: T_{shaft} = torque at shaft

η = efficiency

i = gear ratio

T_m = torque motor

T_{sf} = static friction

f. Give the relationship between the speed of the shaft in [rpm] and the encoder frequency in [Hz].

2 Hardware design motor controller

Figure 1.1 shows the block-scheme of the electronics of the motor controller. It consists of the following blocks:

1. A full bridge-converter (SC1). This circuit converts the input voltage to a positive or negative adjustable voltage to control the motor speed and direction. A shunt resistor is included in the motor current circuit to measure the motor current.
2. A shunt resistor (R1), current sensor circuit (SC4) and voltage sensor circuit (SC5)
3. A 5V supply circuit (SC6)
4. A shield to connect to an ESP32 MCU board (SC2)
5. A connection circuit for the motor/encoder (SC8)
6. A connection circuit for the gyroscope MPU6050 (SC9)
7. A buzzer, an RGB-LED and 3 buttons (SC7)

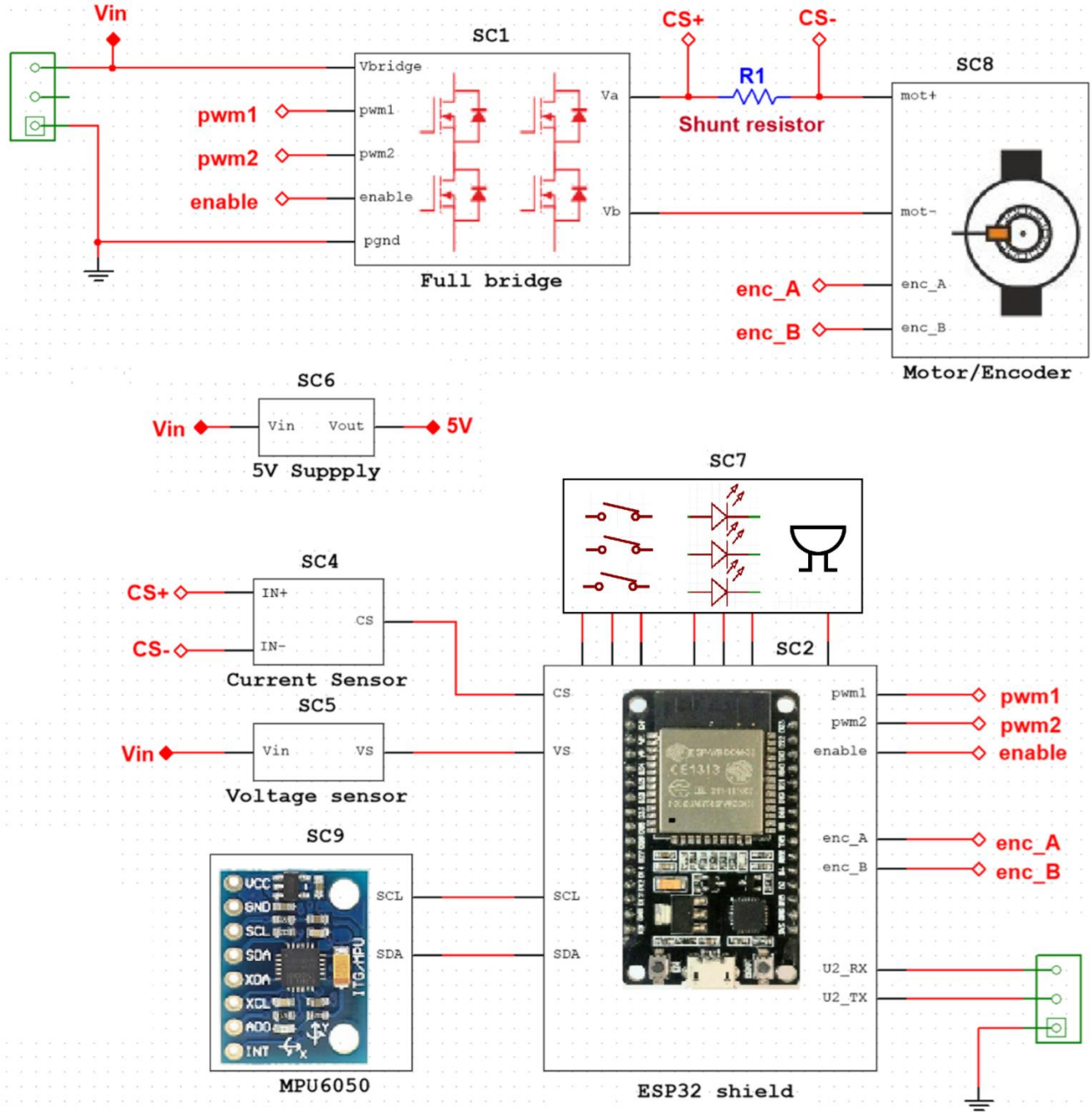


figure 2.1: Block-scheme of the motor controller.

Question 2 (Assessed together with question 11)

Making a PCB of the motor controller involves 2 major activities: First, you have to enter the schematics of the hardware and the components that are used. Then, the PCB design can be made. In this course, we will use Easy-EDA for both the input of the schematics and the PCB-design. Easy-EDA is a free and very easy-to-use tool. Browse to Download Center - EasyEDA. Download and install the latest version of EasyEDA Std Edition.
Enter all the schematics of figure 2.2 up to figure 2.7 in an EasyEDA project. Use the BOM given in Appendix A to select most of the components. Use the supplier part column to find the component (Place → Type supplier part in the LCSC search field).

Question 3 (6 pnts)

Study reader [1] that explains the working of Mosfets and a full bridge circuit. Then, answer the questions below.

Figure 2.2 shows the H-bridge circuit that provides the motor voltage. The base of it consists of 4 mosfets (Q1 up to Q4) and 2 half-bridge drivers (U1 and U2) that provide the gate signals of the 4 mosfets. The table below shows the state of the mosfets and voltage on the output pins as a function of the control signals pwm1, pwm2 and enable.

enable	pwm1	pwm2	Q1	Q2	Q3	Q4	U_{CS+} [V]	U_{mot-} [V]
0	x	x	off	off	off	off	floating	floating
1	0	0	off	on	off	on	0	0
1	1	0	on	off	off	on	Vin	0
1	0	1	off	on	on	off	0	Vin
1	1	1	off	on	on	off	Vin	Vin

x= don't care

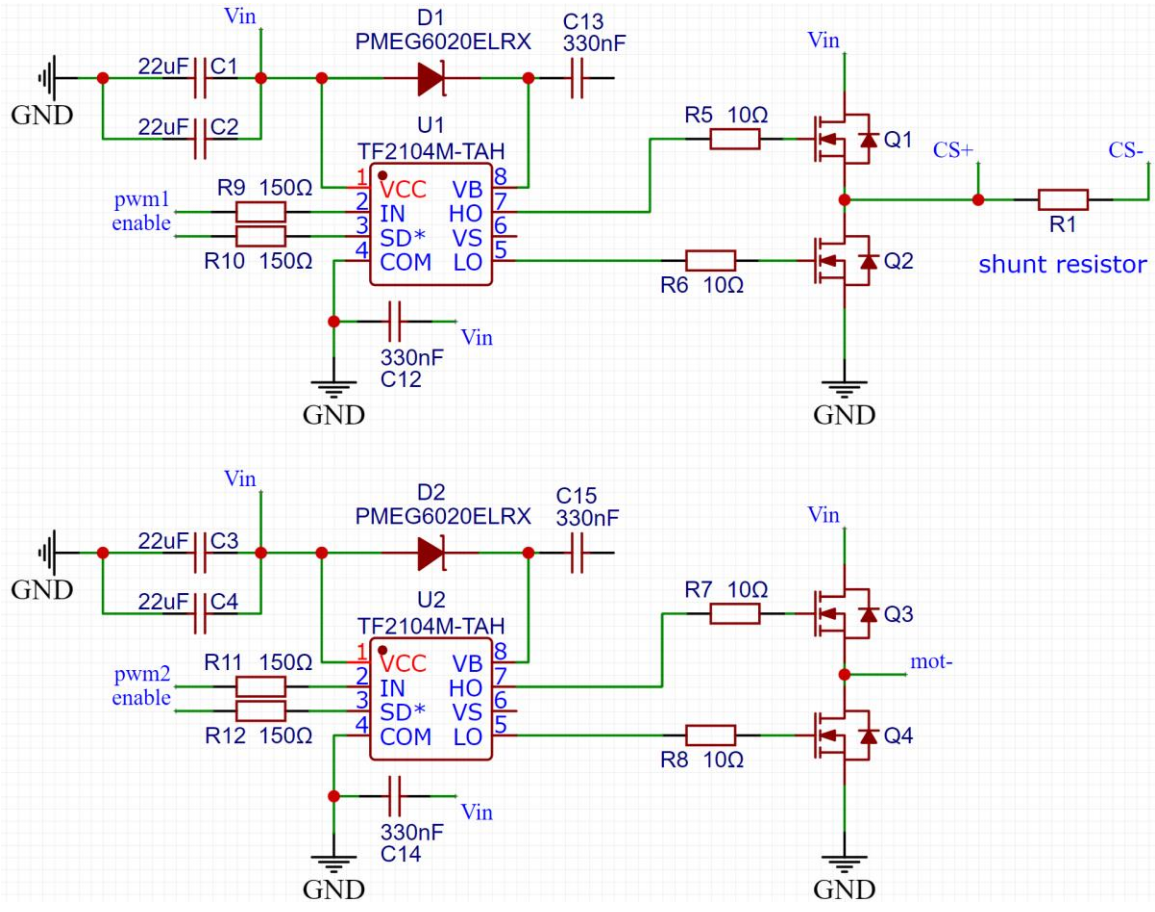


figure 2.2: Full-bridge converter that controls the motor. The connections of C13, C15 and pin Vs of the TF2104 gate drivers are not given. You have to find out yourself how they must be connected.

Assume that the H-bridge is connected to a motor that consumes a 3 [A] constant current. All mosfets may be considered as ideal (R_{on} = 0Ω). The input voltage is Vin = 12V. Control signal enable is 1. Control signals pwm1 and pwm2 are block shaped signal with a switching frequency of 50kHz and an adjustable duty-cycle.

a. Show in a graph the voltage at CS+ as function of the time at a duty-cycle of pwm1 of 66.7%.

b. Give the relation between the average voltage on CS+ as a function of the duty-cycle of pwm1.

c. Show in a graph the current through all 4 mosfets as a function of the time at pwm1 = 66.7%.

d. Find a suited mosfet for this application. The following constraints apply:
- the threshold voltage is at most 4[V]
- the mosfet must have a SMD package.
- the absolute maximum ratings of the drain source voltage is at least 30V (twice as much as needed)
- the continuous drain current is at least 6A (twice as much as needed)
- the mosfet has a good quality / price ratio (cheap and low power losses).
Give the main characteristics and a link to the datasheets in the answer document.

e. Calculate the power losses and efficiency of the full bridge at a duty-cycle of pwm of 66.7% (consult [2] for the calculation of power losses). Use the characteristics of the mosfet that you have chosen in question 3d.

f. What is the purpose of capacitor C13 and C15? What should the right-hand connector pin be connected to (see Figure 2.2)? And what should pin VS of the half-bridge driver TF2104 be connected to?

Question 4 (3 pnts)

Figure 2.3 shows the current sensor circuit. The base of the current sensor is a INA181 bidirectional current-sense amplifier. This chip measures the voltage across the shunt-resistor (R1) that is placed in series with the motor (see figure 1.1). The left part of the circuit around opamp U7A generates a constant reference voltage Vref that is needed by the INA181.

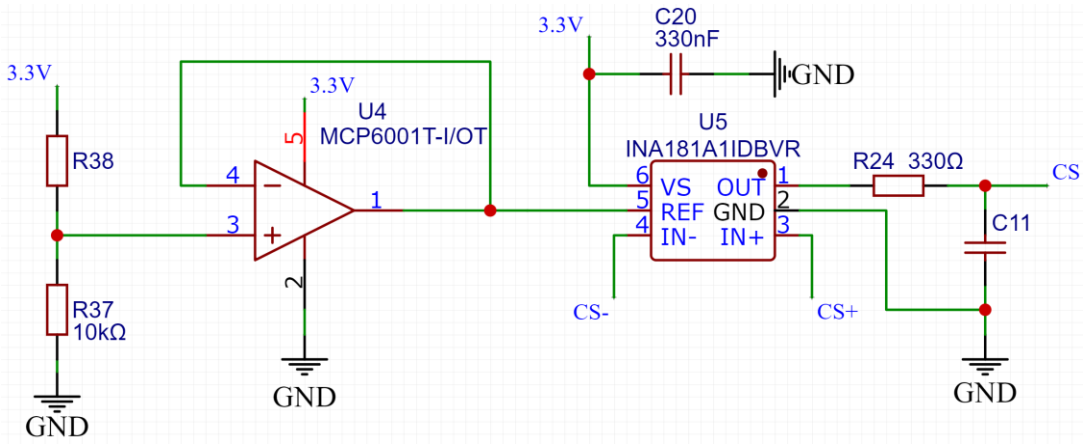


figure 2.3: Current sensor circuit.

a. Find in the datasheets of the INA181A1IDBVT the relationship between the voltage on the output pin OUT and the voltage on the inputs pins IN+, IN- and REF.

b. Find the value of R38 and the value of R1 (shunt-resistor, see figure 1.1), so that we have:

motor-current = -3A → VCS = 0.3V (V_{CS} = output voltage of current sensor)
motor-current = +3A → VCS = 3.0V

c. Because of the switching nature of the H-bridge, the motor current has a ripple. Calculate C11 so that the ripple on the motor current is attenuated by a factor of 100. Assume a switching frequency of 50 [kHz].

Question 5 (4 pnts)

Figure 2.4 shows the voltage sensor circuit. It consists of a combined attenuation circuit and a low-pass filter at the input, an opamp buffer-circuit and a low-pass filter at the output. Literature [3] explains the working of the opamp circuit.

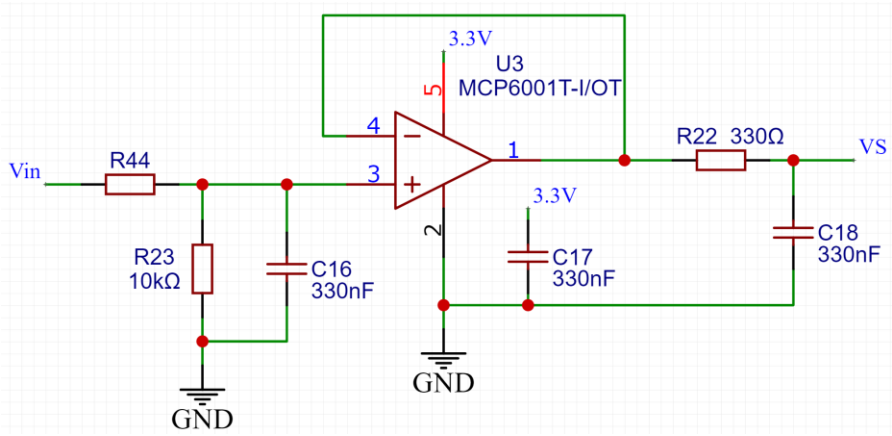


figure 2.4: Voltage sensor circuit.

a. Calculate the value of R44 in order to get an attenuation factor of 5. In that case, the maximum input voltage of Vin=15V results in VS=3V, which is the close to the maximum ADC input voltage of the ESP32.

b. Calculate the cut-off frequencies of the input and output low-pass filters.

c. Calculate the inaccuracy of the voltage sensor in percentage at an input voltage of Vin = 12V at 85°C. Assume a 1% tolerance of the resistors and 10% tolerance of the capacitors. Find the specifications of the opamp in the datasheets and consider the error caused by the input bias current (the current that flows into the inputs of the opamp), the open-loop gain A_0 and the offset voltage.

Remark: The output voltage of an opamp that has an offset is given by: $V_{out} = A_0 \cdot (V_+ - V_- - V_{offset})$

Question 6 (2 pnts)

Figure 2.5 shows the 5V-voltage supply circuit. This 5V-circuit provides the supply voltage of the ESP32 board and the gate drive circuits. The base of this circuit is a LMR14050S buck-voltage regulator (see literature [2]).

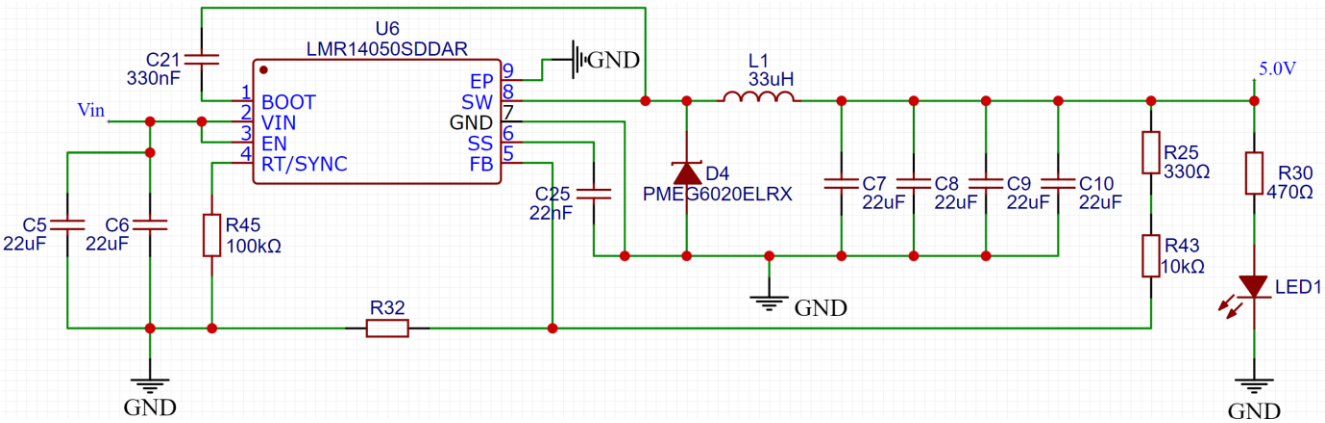


figure 2.5: Schematics of the 5V voltage regulator.

a. Calculate the value of resistor R32 to obtain a 5V supply voltage.

b. Calculate the power losses and temperature rise of the LMR14050S at an input voltage of 12V and output current of 0.2A. To do so, you have estimate the efficiency of the LMR14050S from the datasheets. Use a thermal resistance value of 42.5 [°C/W] *(remark: this is Junction-to-ambient thermal resistance).*

Question 7 (1 pnt)

The MPU6050 subcircuit (SC9 in figure 2.1) holds an 8-pins header connector and a few resistors. The resistors are used as pull up resistors of the I2C control signals. Explain what a pull up resistor is and how much Ohm is typically is used for I2C. Make a screen dump of the schematics of the MPU6050 sub circuit (SC9) and add it in your report.

Question 8 (2 pnts)

Figure 2.6 shows the user interface of the motor controller consisting of a RGB-Led, 3 push buttons and a buzzer.

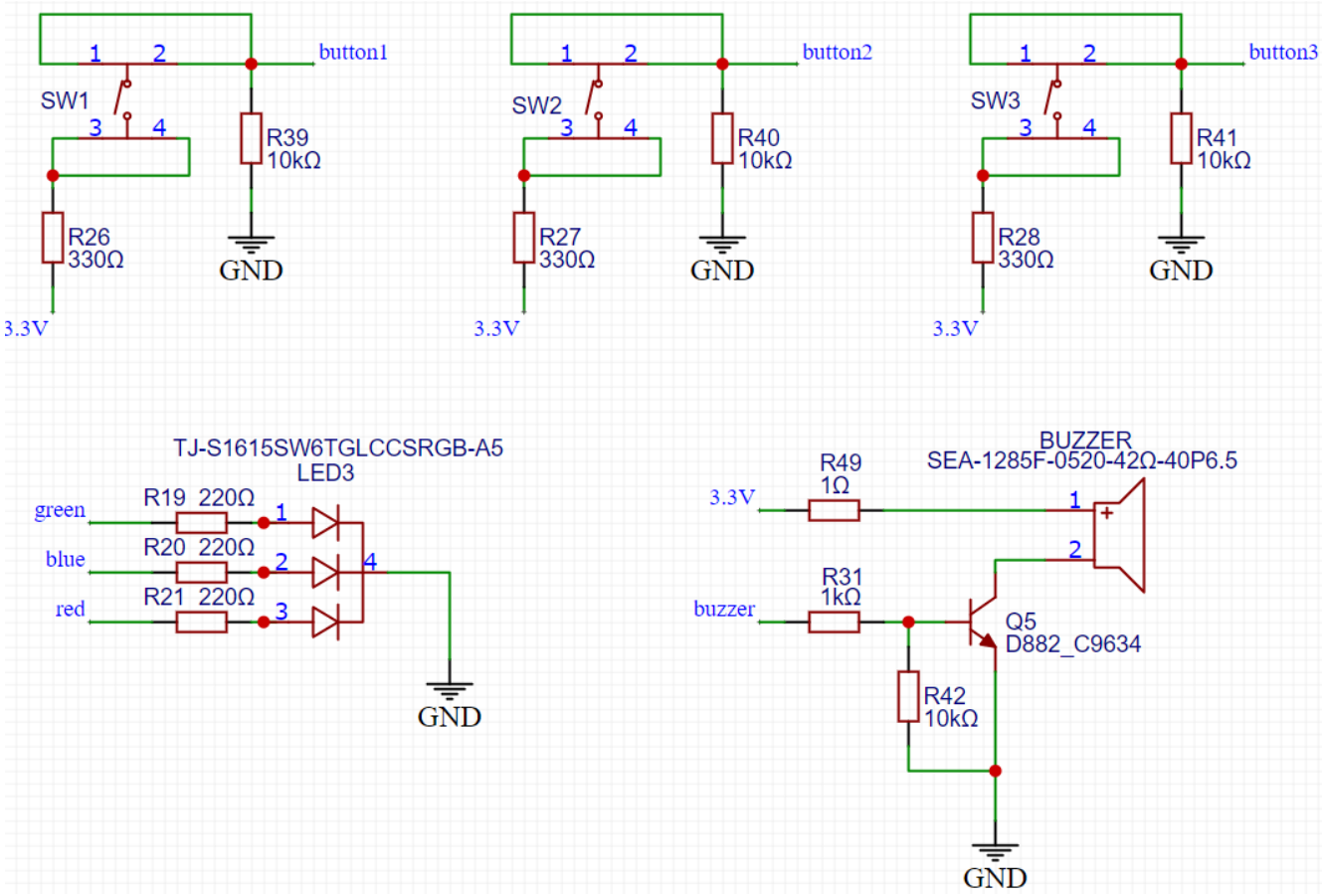


figure 2.6: Schematics of the user interface electronics.

a. Calculate the total current consumption of the RGB LED if all 3 leds are turned on.

b. Explain why it is necessary to use a transistor to provide the buzzer current. Do this by comparing the maximum current of a GPIO pin of the ESP32 to the buzzer current at 3.3V.

Question 9 (2 pnts)

To control all electronics of the motor controller, a JOY-IT ESP32 board is used (DOIT ESP32 DEVKIT 1). This board houses a ESP32 microcontroller, which is a 160MHz - dual core 32-bit microcontroller which has a 2.4 GHz WiFi / bluetooth module, 512 kB SRAM, 4 MB memory, 2x DAC, 15x ADC, 1x SPI, 1x I²C, 2x UART.

The board has two 15 pins headers for interfacing external electronics. You can find the pin layout https://lastminuteengineers.com/esp32-pinout-reference/. To make the motor controller work, we have to connect the signals of the schematics discussed in the previous sections to the 15 pins headers of the ESP-32 board (see figure 2.7).

Specify which motor controller signals should be connected to which pins of the ESP-32 board. The table below describes in the first 2 columns the signals that are used in the motor controller design.

Fill out the last 3 columns completely.

Remark: There are many possibilities for the connecting of the electronics.

Signal	Description of signal	GPIO-pin ESP32	Connector	Pin
5.0V	Plus of 5V power supply (connected via diode, see figure 2.6)	-	J1_ESP32	1
3.3V	Plus of 3.3V power supply	-	J2_ESP32	1
Ground	Power supply (minus)	-	J1_ESP32	2
Ground	Power supply (minus)	-	J2_ESP32	2
RX2	Receive pin serial port connection ESP32 with high level controller	GPIO16	J2_ESP32	6
TX2	Transmit pin serial port connection ESP32 with high level controller			
SCL	I2C clock signal (communication bus with module IMU MPU6050)			
SDA	I2C data signal (communication bus with module IMU MPU6050)			
enc_A	Encoder output A			
enc_B	Encoder output B			
pwm1	PWM signal of half bridge 1			
pwm2	PWM signal of half bridge 2			
enable	Enable signal of both half bridges			
CS	Analogue output voltage of current sensor			
VS	Analogue output voltage of voltage sensor			
button1	Push button 1			
button2	Push button 2			
button3	Push button 3			
green	Green LED of RGB LED			
red	Red LED of RGB LED			
blue	Blue LED of RGB LED			
buzzer	Signal that controls a buzzer			

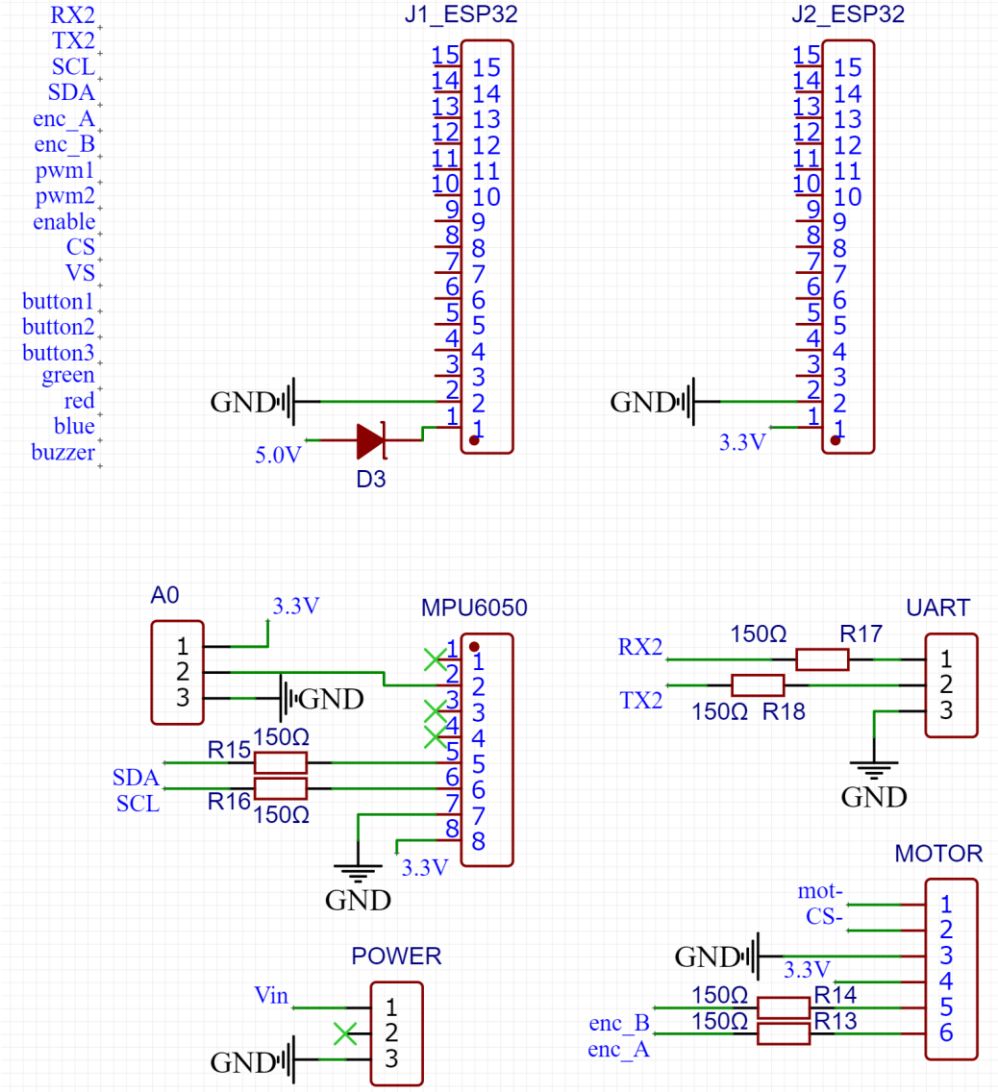


figure 2.7: The motor controller has the following connectors on it:

- *J1_ESP32 and J2_ESP32: Two 15-pins female header to connect to the ESP32-board*
- *MPU6050: An 8-pin female header to connect to the MPU6050 board*
- *A0: A 3-pin male header to connect the address-pin A0 of the MPU6050 board to either 0V or 3.3V*
- *UART: A 3-pin male header to connect the ESP32 board to a serial port*
- *POWER: A 3-pin male header to connect the motor controller board to a power supply.*
- *MOTOR: A 6 pin male header to connect the motor controller board to the RB-Dfr-444 motor/encoder.*

Question 10 (1 pnt)

All modern electronic circuits nowadays have *decoupling capacitors*.

a. Explain what that means and why they are used.

b. List all decoupling capacitors of the schematics shown in figure 2.2 up to 2.7.

3 PCB design

Question 11 (20 pnts)

Make the PCB-design. Follow the instructions below. When finished, add a screen dump of the PCB-design in the answer document that shows all important layers. A zip-file that holds a folder with the easyEDA project, the BOM, Ferber files and the Pick and Place file must be turned in later in Handin.

Instructions for making a PCB

Once the schematics is finished, you can start designing the PCB. To do so, open the schematics and select Design → Update PCB (or type the key combination <ALT> + U).
When the schematics has errors, a warning window opens (see figure 3.1). If this windows appears, click the 'Yes, Check Nets' button to see what is wrong and correct it.

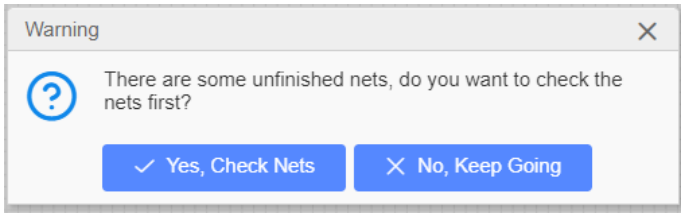


figure 3.1: Error window that appears when there is an error in the schematics while it is being transferred to the PCB-editor.

When there are no errors, EasyEDA will ask you to create a PCB-design in the current project. Select 'Yes'. Then select a 2 copper layer PCB, a rectangular PCB-outline and enter the PCB-dimensions (80x120mm). Click 'OK'. Then, the PCB design workspace opens which looks something like figure 3.2.

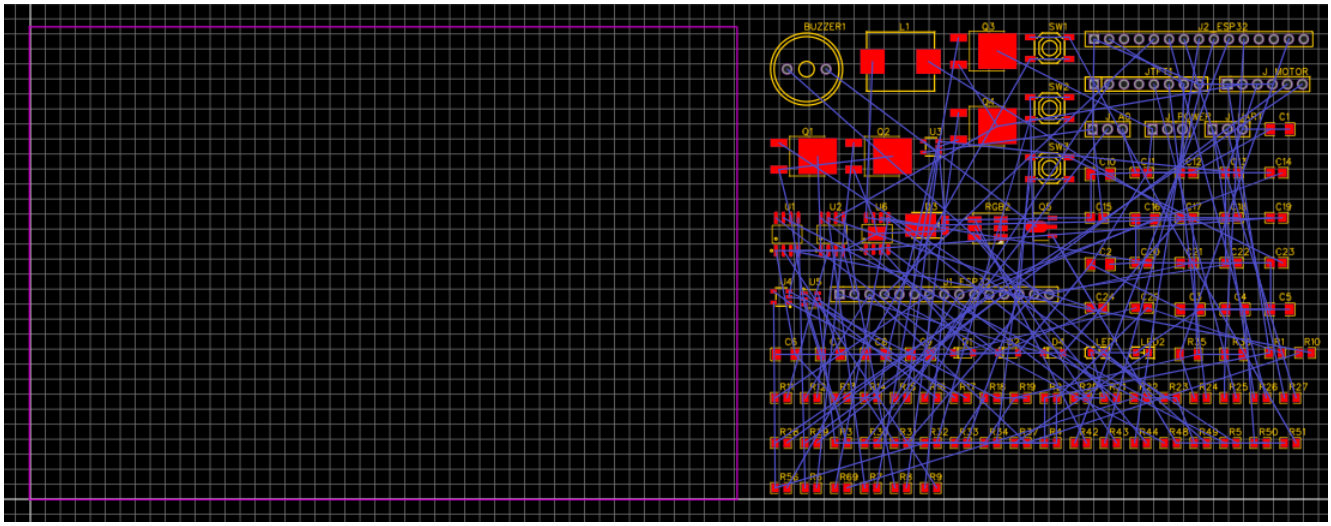


figure 3.2: EasyEDA PCB-design workspace.

The purple rectangle on the left of figure 3.2 is the PCB-outline. All components and connections between them (traces) must be placed within this rectangle. At the right part of figure 3.2 you see the footprint of all components. The blue lines are the called the ratsnest and represent the connections between the components. Designing a PCB involves basically placing all components within the PCB-outline and connecting them.
Before making the PCB, it is advised to follow the PCB-design tutorials mentioned in the links below.

PCB Basics - SparkFun Learn
How to Design a PCB Layout: A Comprehensive Guide (wevolver.com)

Below, you have some guidelines that define the quality of the PCB.

1. Use as much as possible, the top copper side of the PCB for components and traces and the bottom copper side for a power plane connected to the ground. The use of power planes is in general widely applied in PCB's because of the following reasons:
- A power plane has a much lower impedance than normal traces at high frequencies. Especially for the ground it is important that the impedance is low, otherwise potential differences in the ground net may arise.
- You can easily connect ground connections with the power plane by using vias.
- Power planes reduce EMI (Electro Magnetic Interference) because they 'absorb' electromagnetic radiation.
Because the bottom copper layer is used as a ground power plane, you must try to place all traces, except the ground traces, on the top copper side. To avoid crossing traces, you might have to rotate or relocate components on the PCB. Sometimes you can avoid crossing traces by routing traces in between the pads of components. If it is really impossible to place a trace at the top side without crossing other traces, you can place (a part of) the trace on the bottom side. Make this bottom side trace as small as possible. This reduces the disruption of the power plane by the trace. Figure 4.4 shows an example of a small part of a PCB design.

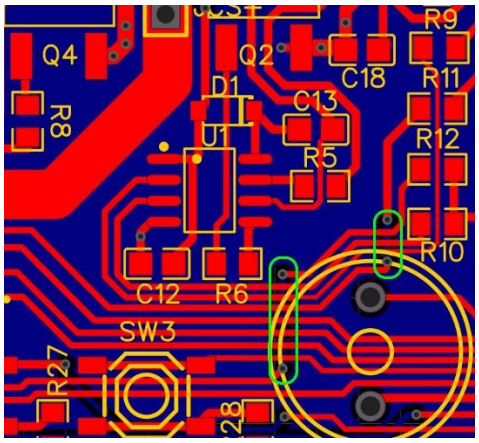
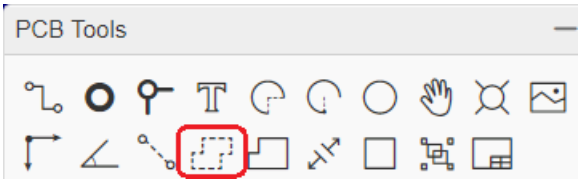


figure 4.4: Design of a PCB. The red squares are the pads of the components on the top side, the red lines are the traces on the top side that connect the pads of the components to each other. The blue background is the ground copper plane on the bottom side. The small grey circles with a black center point are vias. They connect the top copper with the bottom copper. Most of them are used to make connection of components with the ground. Enclosed by green curved rectangles, there are small traces on the bottom side. Note that some traces lie in between pads of components..

Place the power plane as follows:
- Select the Solid region from the PCB Tools.
- Click on the four corners of the PCB outlines. Right click the copper area. Select BottomLayer. Type GND at the Net input field. Enter a clearance of 0.32mm.



2. Start placing the connectors first. The ESP32-connectors (J1_ESP32 and J2_ESP32) must be placed exactly according the spacing of the on the JOY-IT ESP32 board (see: NodeMCU ESP32 - Joy-IT). Ensure yourself that the headers are correctly oriented. Lock the position after placing them so they cannot be moved accidentally..

Step 3. Start placing the components of the power circuit, because they may cause a lot of disturbances if they are not located well. Drag the components of the half-bridges (figure 2.2) into the workspace. Place the components close to each other in a logical way and connect them to each other. Use a trace width of at least 3mm. Then, place the gate drivers of the half-bridges in the workspace close to the mosfets and connect them. Place all decoupling capacitors close to the mosfets and gate drivers.

Step 4. Then place the components of the other sub circuits. Place the decoupling capacitor of each IC close to the IC.

Step 5. Finally, carry out a design rule check (Design → Check DRC). Correct possible errors. Export and save the BOM-list, Gerber files and Pick and Place file (they must be turned in later).

4 Unit testing of PCB and firmware

Question 12 (2 pnts)

Mount the resistors R1, R32, R38 and R44 on the PCB. You can get them at the electronic components warehouse (room H0.32 at the first floor). Ask for an appropriate resistor in the E12-series in a 1206 package. The PCB has 4 resistor footprints in parallel to create the desired value of R1 (they are labelled R1 up to R4).
The link How to Solder SMD Resistors using Soldering Iron Quickly (youtube.com) shows how to solder them on the PCB.
Make a good picture of the PCB after soldering so that the quality of the soldering can be assessed. Add it in the answer document.

4.1 Testing the 5V power supply

When designing an embedded system like the motor controller, it is very important to be sure that the hardware works well and as expected. For this PCB, there is no absolute guarantee that the electronics works well. The design is new and could have errors. Also, you there can be a mistake in the calculations of the resistors R1, R32, R38 and R44 that may cause malfunctioning. Therefore, you have to test the electronics.

Question 13 (4 pnts)

a. At first, we test the 5V-supply. It is very important that you test if it works, because damage to all other electronics may arise if the power supply doesn't work well.

To do so, you have to measure the 5V supply voltage as a function of the input voltage in a range from 6V to 15V. This measurement is carried twice: once if the 5V-supply is unloaded and once at a 10Ω load resistor.

For this measurement, you need 2 multimeters and an adjustable power supply. Before starting the measurement, adjust the current limit of the power supply to a relatively low value (about 0.1A) and the voltage to 9V. Then, turn the adjustable supply off and connect it to header POWER on the PCB via a DC-current meter (use one of the multimeters). Connect the second multimeter in the DC-voltage measure mode to header J5V on the PCB (see figure 4.1). Don't connect the 10Ω load resistor yet.

Turn on the adjustable power supply again. Check if LED1 lights up and if the current limit of the power supply current doesn't invoke. If true, you can start measuring. Otherwise, there is something wrong on the PCB and you have to find the error.

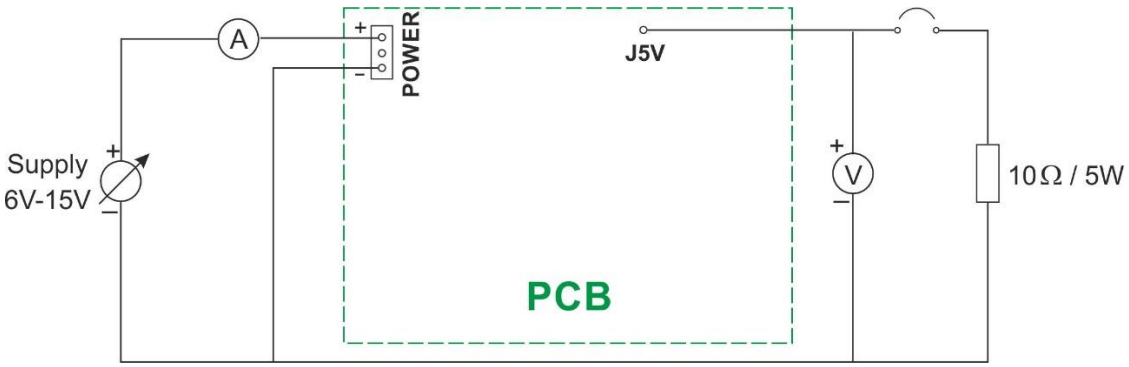


figure 4.1: Measurement setup for testing the 5V-supply circuit.

Adjust the current limit of the power supply to 1 [A]. Measure the supply current and the voltage on pin J5V as a function of the input voltage. Do this in a range from 6V to 15V in steps of about 1V. Carry out the measurement unloaded and loaded. Use the other multimeter to measure the voltage on pin J5V and the supply voltage.

Fill in the table below.

Supply voltage [V] (unloaded)										
Voltage on J5V [V] (unloaded)										
Supply current [mA] (unloaded)										
Supply voltage [V] (loaded)										
Voltage on J5V [V] (loaded)										
Supply current [mA] (loaded)										

b. Make a Python script to create a graph of the voltage on the J5V pin and the efficiency of the 5V-supply as a function of the input voltage for the loaded situation. Explain if the results match with what can be expected from the datasheets of the LMR14050SDDAR buck converter. Add the graph and the script in your answer document.
Hint: "Chapter 5 of the reader "Introduction Course Programming in Python.pdf" explains how to make graphs in Python using the libraries numpy and matplotlib.

4.2 Testing the ESP32 board and the user electronics

The second test we carry out is the testing of the ESP32 board. We have to get this board working before we can test other circuits because the ESP23 board provides the 3.3V supply for the voltage and current sensor and it provides and processes all data signals to and from the other electronic circuits.

We test the ESP32 board in 2 phases: First of all, we test the software development environment. We use Micro-Python as programming language and Thonny as IDE (Integrated development environment). Follow the instructions in Appendix B to install Thonny and the MicroPython firmware.

Question 14 (4 pnts)

Disconnect the USB-cable from the ESP32 board and plug it into the PCB motor controller board while it is powerless. *Nota bene: In general you should never plug an electronic board into another board if one of them is under voltage! This might damage the boards.*

Connect the ESP32 board to your laptop again and check if the flashlight program is still working. If not, there is something wrong with the PCB. In that case, warn the supervisors.

Write an application with the following functionality.

Pressing push button 1 (SW1) will alternately turn on and off the red RGB led

Pressing push button 2 (SW2) will alternately turn on and off the green RGB led

Pressing push button 3 (SW3) will alternately turn on and off the blue RGB led

When all three RGB LEDs are turned on, the buzzer will emit a siren sound consisting of alternating tones of 375Hz and 500Hz with a duration of 400ms each.

There may no delays in the processing of the button presses. So, if the user presses push a button, immediately after the corresponding led must toggle. The table below shows the GPIO connections.

Red RGB	Green RGB	Blue RGB	Push button 1 SW1	Push button 2 SW2	Push button 3 SW3	Buzzer
GPIO 19	GPIO 5	GPIO 18	GPIO 39	GPIO 36	GPIO 15	GPIO 4

Include the boot.py and the main.py in your answer document.

Hint 1: To create a sound with the buzzer, you have to define the buzzer pin as a PWM pin. Consult Quick reference for the ESP32 — MicroPython 1.16 documentation for more information about using PWM.

Hint 2: Don't use the delay() function. Use interrupts to process button presses, because that doesn't give a delay. Refer to the links below to understand how this works.
MicroPython: Interrupts with ESP32 and ESP8266 | Random Nerd Tutorials
Interrupts ESP32 MicroPython : React to events (upesy.com)

4.3 Testing the voltage sensor

To test the voltage sensor circuit and the ADC of the ESP32 that converts the sensor voltage to a digital value, we use the circuit given in figure 4.2. The output of the voltage sensor VS is connected to pin 9 of the left ES32-board connector.

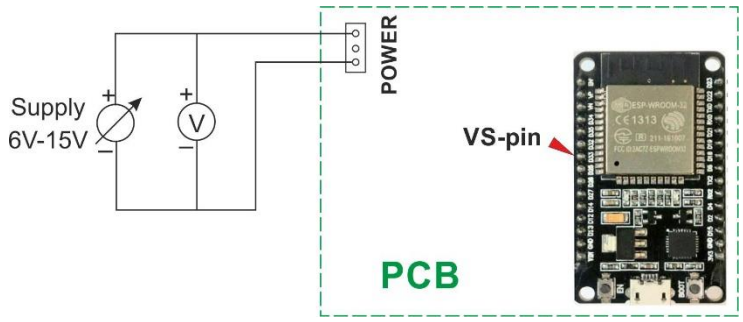


figure 4.2: Measuring setup for voltage sensor testing. The voltages Vin and VS are measured with a test probe.

To prevent strange behavior of the half bridges, you best disable them by keeping the enable signal of the half bridges low. The enable signal is connected to GPIO13 of the ESP32 board. LED2 on the PCB is also connected to this pin, so you can see whether the enable pin is high or low.

Question 15 (4 pnts)

Add in boot.py of question 14 code to configure pin GPIO13 as an output pin. Set it to 0V to disable the half bridges. Also, add code in boot.py to configure pins GPIO32 and GPIO33 as analog input pins. GPIO32 is connected to CS and GPIO33 is connected to VS. Set the attenuation of both channels to 11dB to measure in a full range from almost 0V to 3.3V. Set the ADC-width to 12 bits. Give recognizable names for the ADC objects that refer to GPIO32 and GPIO33.
Remark: See Quick reference for the ESP32 — MicroPython 1.15 documentation for more info about the ADC.
Create in a new main.py the code to print in an endless loop each 500 [ms] the voltage on the VS pin in [mV]. Use the method read_uv() because it is much more accurate than the read() method.

a. Build the measurement circuit of figure 5.2. Measure Vin with a multimeter and VS with the ADC on the ESP32. Carry out the measurement in a range of 6V ≤ Vin ≤ 15V in steps of about 1V. Fill in the table below. Check if the measured values of VS match your expectations. If not, try to find out what goes wrong.

Vin [V] (multimeter)										
VS [mV] (ESP32)										

b. Make a Python script that makes a plot of VS in [mV] as a function of Vin in [V]. Show grid lines.

c. Theoretically, the voltage on the VS-pin is proportional to Vin because VS is obtained from Vin via a resistor divider that consists of the resistors R23 and R44. So, it applies:

$Vin = c_1 \cdot VS$

Extend the python script of question 15b to find the ratio factor c_1 by averaging the ratios Vin/VS of each measurement. Compare this value to what can be expected from the resistor values of R23 and R44. How much is the difference in percentage? Explain a difference of more than 3%.
Add the python script in your answer document.

4.4 Testing the current sensor

The circuits given in figure 4.3 are used to test the current sensor circuit,. The left figure is used to measure at a positive current (a current from CS+ to CS-), the right figure is used to measure at a negative current (a current from CS- to CS+). The current is measured with a multimeter, the voltage on the current sensor pin (CS-pin) is measured with the esp32. The supply is an adjustable power supply of at least 3A.

Question 16 (4 pnts)

a. Change main.py of question 15 in such a way that each 500 [ms] the voltage on the CS-pin is shown.
Build first the left measurement circuit of figure 4.3. Increase the current in steps of about 0.3[A] from 0[A] to 2.4[A]. Fill in the table below. Check if the measured value of CS and the ADC value match your expectations. If not, try to find out what goes wrong.

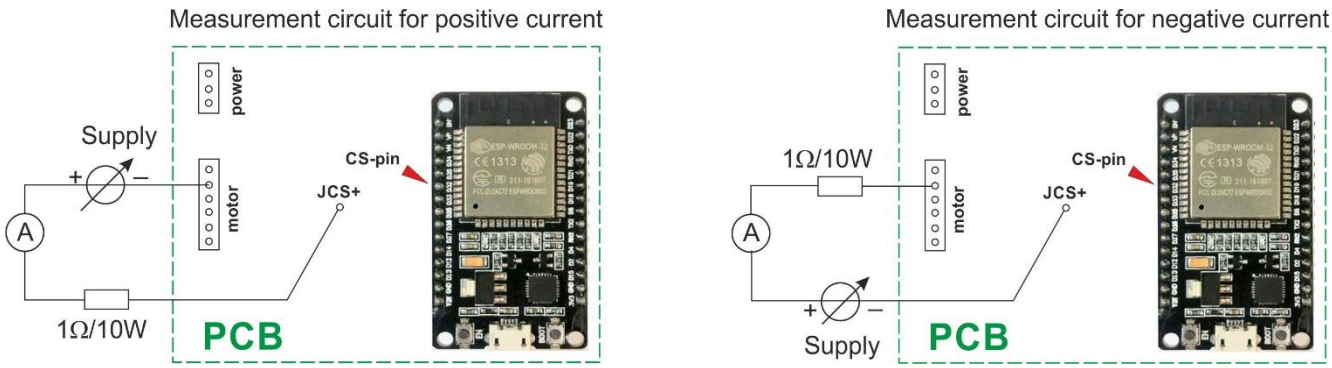


figure 4.3: Measurement setup for testing the current sensor.

Measurement results for positive current.

I [mA] (multimeter)	0								
V_{CS} [V] (ESP32)									

Repeat the measurement for negative currents. Decrease the current in steps of -0.3[A] from 0[A] to -2.4[A].

Measurement results for negative current.

I [mA] (multimeter)	0								
V_{CS} [V] (ESP32)									

b. Make a Python script that makes a plot of the voltage V_{CS} in [mV] as a function of the current in [mA]. Show grid lines.

c. Theoretically, the current sensor voltage is linear with the current (see question 5). It applies:

$$I = I_0 + c_2 \cdot V_{CS}$$

Extend the python script of question 16b to find the constants I_0 and c_2 via linear regression. Use the numpy function polyfit() for this. Compare the values of I_0 and c_2 with what can be expected based on the chosen values of R1 and R38. How much is the difference in percentage? Explain a difference of more than 3%.
Add the python script in your answer document.

4.5 Testing the H-bridge converter

To test the H-bridge converter, we use the circuit given in figure 4.4. The outputs of both half bridges are connected to a scope, so we can see their characteristics (amplitude, frequency and duty-cycle).

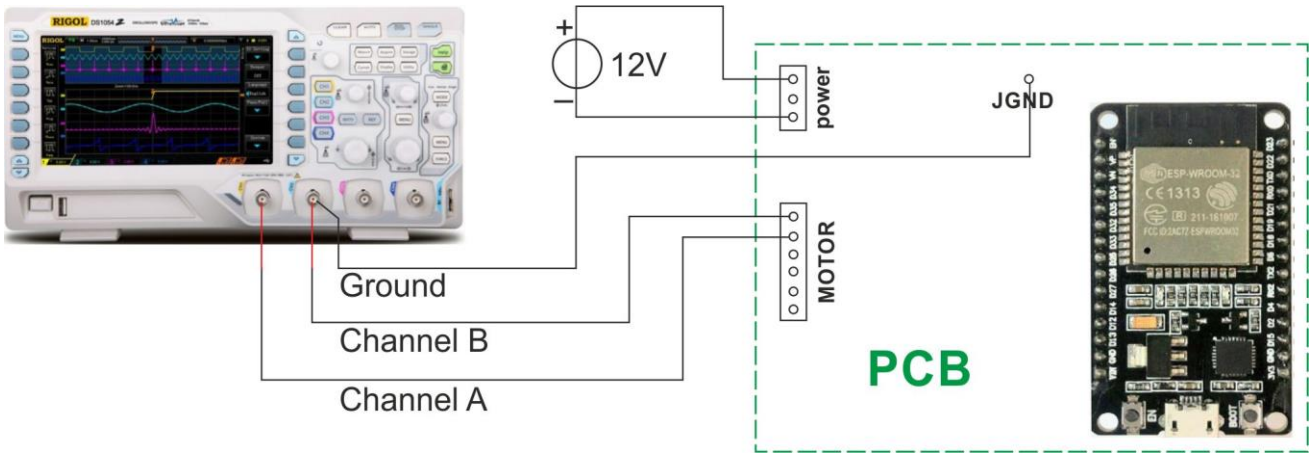


figure 4.4: Measurement setup for testing the full bridge converter.

Question 17 (3 pnts)

a. Extend the python script boot.py of question 16 with the configuration of GPIO 25 and GPIO 26 as PWM-pins. GPIO 25 is connected to signal pwm1 (see figure 2.2), GPIO 26 is connected to signal pwm2. Adjust the frequency to 50 [kHz] and the initial duty-cycle to 0 of both pwm-signals.

Write a new script main.py that first enables GPIO 13 (enable pin of the half bridge drivers). Then, it asks the user in an endless loop to enter a duty cycle in a range from -950 to 950. If the input is positive, the software sets the duty-cycle of pwm1 to the entered value and the duty-cycle of pwm2 to 0. If it is negative, it sets the duty-cycle of pwm1 to 0 and the duty-cycle of pwm2 to the absolute value of the entered value (see figure 4.5). Use the python function int() to convert the string entered by the user to a numerical integer value. Add the code of boot.py and main.py to your answer document.

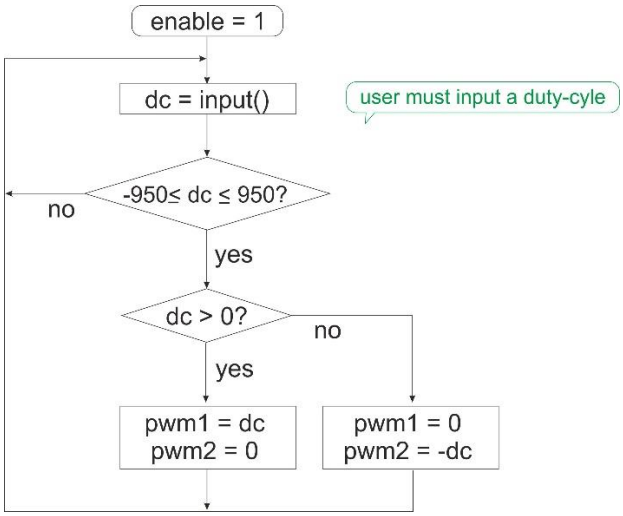


figure 4.5: Flow chart of main.py script.

Build the measurement circuit of figure 4.5. Measure the frequency and duty-cycle of the half bridge outputs with the measure function of a scope at the following settings:

	Frequency scope- voltage [kHz]	Duty-cycle bridge [%]	Make scope image
pwm1 = 300, pwm2 = 0			Yes
pwm1 = 600, pwm2 = 0			No
pwm1 = 900, pwm2 = 0			No
pwm1 = 0, pwm2 = 300			No
pwm1 = 0, pwm2 = 600			Yes
pwm1 = 0, pwm2 = 900			No

Add scope images with one period visible to your answer document for the situations where pwm1=300 and pwm2=600.

Remark: If the scope image doesn't show the expected wave form, check the following:

- Is the scope is working correctly? Connect the scope channels to 12V. Do you see the scope lines at 12V?

- Is the ESP32 working correctly: measure the voltage on pins 7 and 8 with a test probe. Do you see a pwm signal?

- Are the signals on the pins of the level shifter and gate drivers correct?

b. Find out experimentally what the maximum duty-cycle is. Do this by increasing the range of the duty-cycle input of the user to check -1024 ≤ dc ≤ 1024. Then increase the duty-cycles in steps to the value until the output signal stops showing a block-shaped signal or exhibits irregular behavior. Find the maximum duty-cycle of both half-bridges.

4.6 Testing the motor/encoder

Question 18 (4 pnts)

Extend the functionality of main.py of question 17 with the showing of the encoder frequency and the motor current if the user presses only the enter-button when he is asked to enter the duty-cycle (see flow chart below). Appendix C explains how to read the encoder frequency. The encoder signals encA and encB are connected to gpio pins 34 and 35 of the ESP32. The file Encoder.py should be on the ESP32 (see Appendix B).

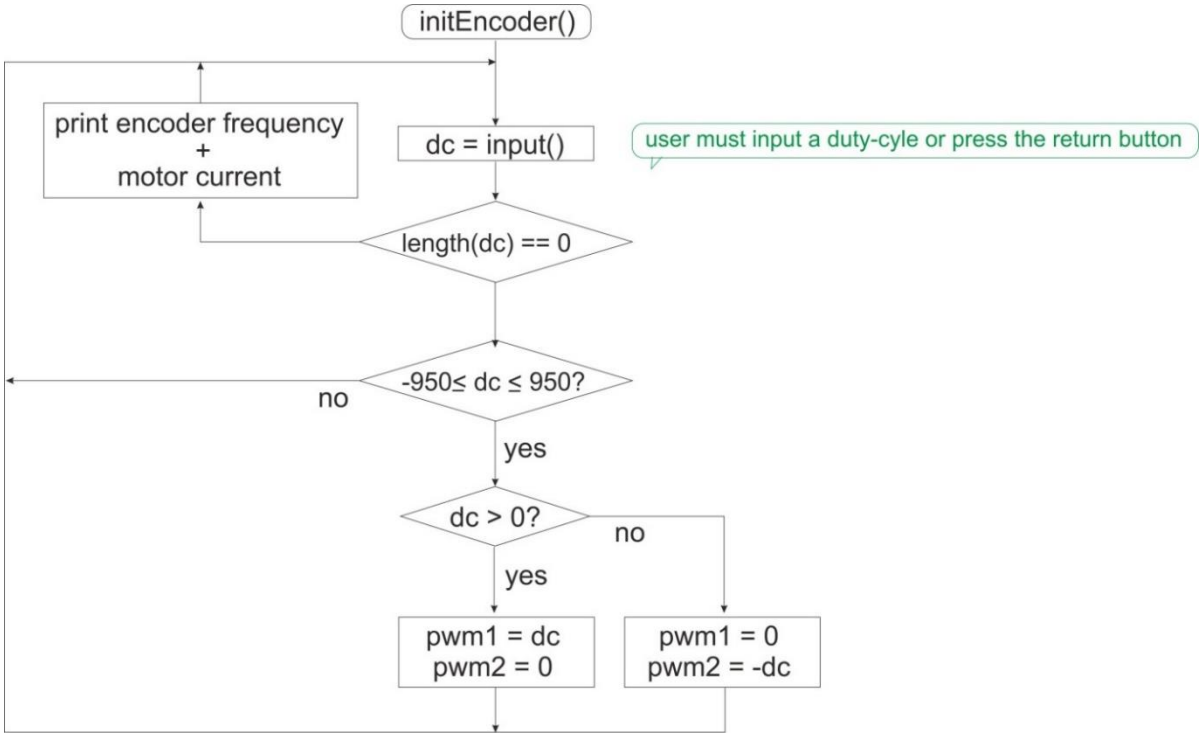


figure 4.6: Flow chart of main.py script.

The motor current can be determined by reading the ADC value of the current sensor and the equation $I = I_0 + c_2 * V_{CS}$ found in question 16c. Appendix C explains how to read the encoder frequency.

Connect the motor/encoder to the motor header on the PCB-board. Connect the red wire of the motor/encoder to the header pin labelled “rood”.

Note bene: Don’t connect the motor/encoder in the wrong way, you might blow up the encoder!!!

Adjust the battery voltage (Vin) to 12V. Test if the motor/encoder works by increasing the duty-cycle in small steps. Fill in the table below and include it in your answer document.

Half bridge 1 active (pwm2 = 0)			Half bridge 2 active (pwm1 = 0)		
duty-cycle	I [A]	Encoder frequency [Hz]	duty-cycle	I[A]	Encoder frequency [Hz]
0			0		
100			-100		
200			-200		
300			-300		
400			-400		
500			-500		
600			-600		
700			-700		
800			-800		
900			-900		

b. Write a python script that determines the torque constant and armature resistance of the motor via linear regression using all the measured datapoints from the table above. Compare them to the answers on question 1c and 1d. Add the python script in your answer document.

Hint: The relation between motor voltage and angular speed is given as:

$U_{motor} = R_a \cdot I + k_m \cdot \omega \rightarrow U_{motor} / \omega = k_m + R_a \cdot I / \omega.$

If we define: $y = U_{motor} / \omega$ and $x = I / \omega$, then we have: $y = k_m + R_a \cdot x$

From the measurements, remove the points where f=0Hz. You then can create 18 data points of (x, y) values. Use numpy function polyfit to find k_m and R_a via linear regression.

The motor voltage is not measured but can be determined from the battery voltage and duty-cycle.

The angular speed can be determined from the measured frequency (see question 1f)

4.7 Testing the MPU6050

As specified, the motor controller must have a tilt protection. To measure the tilt, we use a MPU6050 gyroscope/accelerometer board that measures the acceleration in 3 directions. If we assume that the MPU6050 board is standing still, the only acceleration force acting on it is the earth gravitation (9.81 m/s^2) which always works in the vertical direction (z-direction). If the MPU6050 board is aligned exactly vertical, one of the by the MPU6050 measured accelerations will be 9.81m/s^2 and the other 2 accelerations will be 0 m/s^2. If the MPU6050 board is tilted, the measured acceleration which was originally 9.81 m/s2 will decrease and one or both of the others will indicate a non-zero value depending on which direction the board is tilted.

The MPU6050 board has an I2C interface for communication with external devices. Detailed information about it can be found in the datasheets:

- MPU-6000 and MPU-6050 Product Specification (revision 3.4)
- MPU-6000 and MPU-6050 Register Map and Descriptions (revision 4.2)

Question 19 (4 pnts)

Find in the datasheets of theMPU6050 how I2C works (if necessary, find additional information about I2C). Answer the following questions:

a. Explain the following points related to I2C:
- Which signals are involved with I2C?
- What is an I2C address? Where is it used for? What is the address range of I2C?
- How is the start bit defined and how is the stop bit defined in I2C?
- What is the contents of the first byte of a data transfer in I2C (the byte that is transmitted directly after the start bit)?

b. Show in a timing diagram the signals SDA and SCL when a master writes 1 data byte with a value of 150 to a slave with address 54.

c. What is the I2C address of the MPU6050 on your PCB?

Remark: This depends whether pin 2 of the MPU6050 connector is connected to the ground or to 3.3V (see figure 2.7). Odd student couples must connect this pin to the ground, even student couples must connect it to the 3.3V.

d. How long does it take to read the 3 acceleration forces via a burst read at a baud rate of 400k bit/s.

Question 20 (3 pnts)

a. Extend the python script boot.py of question 18 with the declaration of a SoftI2C object with GPIO21 as SDA pin and GPIO22 as SCL pin. Set the frequency to 400 kHz. Find information about the I2C class on:

class I2C – a two-wire serial protocol — MicroPython 1.16 documentation.

b. To communicate with the MPU6050, an object of the class MPU6050 is used. The class is defined in the script file MPU6050_studentversion.py. This file should be on the ESP32 (see Appendix B). In the constructor of this class, the following code is executed:

```
i2c.writeto(self.addr, bytearray([107, 1]))     # What is this command doing?
```

Explain what happens if this code is executed. Why is it necessary? To find the answer, you must find out what data is stored in register 107 and what the reset value of this register is. This can be found in the MPU6050 register map descriptions.

c. Give the equation to convert the raw 16-bits acceleration data to [m/s^2]. Enter the equation in the method readData() of the class MPU6050 (code line 41)

d. Give the equation to calculate the tilt angle [°]. Enter the equation in the method tiltAngle() of the class MPU6050 (code line 47)

e. Define in a new main.py an object of the MU6050 class. Then, each time the user presses the enter key, the acceleration in the 3 directions and the tilt angle are shown on the screen.

Measure the accelerations and tilt angle at the angles give in the table below.

Angle MPU6050 board in x-direction [°]	Measured tilt angle [°]	acc[0] [m/s^2]	acc[1] [m/s^2]	acc[2] [m/s^2]
0				
15				
30				
45				
60				
75				
90				
Angle MPU6050 board in y-direction [°]	Measured tilt angle [°]	acc[0] [m/s^2]	acc[1] [m/s^2]	acc[2] [m/s^2]
15				
30				
45				
60				
75				
90				

4.8 Testing of the serial communication

According to the specifications, the motor controller must be able to communicate with a high level controller via serial communication. Serial communication is very widely used for these kind of applications. Almost all microcontrollers have one or more UARTs onboard to support serial communication. A UART is a peripheral (a piece of hardware inside the controller) that deals almost autonomously with the reception and transmission of one or multiple bytes via the serial communication protocol.

The ESP32 has 3 UARTS on board, so it can setup serial communication to 3 different devices. UART1 is used by the USB-to-serial adapter chip (SILABS-CP2102) on the ESP32 board. This chip is used for flashing of programs and communication via the USB-port of the ESP32 board.

For the communication of the motor controller to a high level controller, we use UART2 of the ESP23. The following serial port communication settings are applied:

- baud rate = 115200 bit/s
- 8 data bits
- 1 stop bit
- no parity bit and flow control

Question 21 (2 pnts)

Read the next link about the serial communication protocol: Serial Communication - learn.sparkfun.com.

Answer the questions below.

a. How long does it take to transmit the string “Hello world” via serial communication at the settings given above.

b. Extend the python script boot.py of question 20 with the configuration of UART2. Find information about the UART class on: class UART – duplex serial communication bus — MicroPython 1.16 documentation.
Make an UART object with the following settings: baud-rate = 115200, 8 data bits, 1 stop bit, no parity.
Write in a new main.py a program with the following functionality. Add main.py in the answer document.

If character ‘R’ is received via UART2, then the red RGB LED turns alternately on and off.

If character ‘G’ is received via UART2, then the red RGB LED turns alternately on and off.

If character ‘B’ is received via UART2, then the red RGB LED turns alternately on and off.

If character ‘A’ is received, then the buzzer turns alternately on (at a frequency of 500Hz) and off.

If the user presses switch 1, then the program transmits the text “switch 1 is pressed”

If the user presses switch 2, then the program transmits the text “switch 2 is pressed”

Hint: Define a bytearray to store data that comes in via the serial port. Use this bytearray as argument for the method readinto() of the class UART to check if new data has been received. If the return value of readinto() is None, then no bytes have been received and the program doesn’t have to do anything. If it is not None, than at least one byte has been received. Read the value of the first byte and transmit the correct string as described.

Question 22 (3 pnts)

The motor controller and high level controller exchange a lot of data such as sensor data, actuator data and configuration settings. Appendix E describes the data protocol that is used for the communication between the high level controller and motor controller.

The script file HighLevelController.py can be used to process the read requests and write commands. This script file defines the 64 elements array reg used for exchanging data and the class HighLevelController that is used to communicate with the high level controller. The class HighLevelController should be used as follows.

- The constructor HighLevelController(uart, tcp) has 2 arguments:
  - uart: This is an object of the UART class used for the serial port communication.
  - tcp: This is an object of the TCPHandler class used for communication via a TCP connection. This is discussed in the next paragraph.
- The method readFromUart() reads incoming characters from the uart object. If a valid read request has been received, it calls the method transmitReadResponse() that transmits the requested data via the uart object. If a valid write command has been received, the function stores the data received in array reg at the given address(es).
- The method readFromTCP() is similar to the method readFromUart () only this method uses communication via a TCP connection. This is discussed in the next paragraph.

In this question you have to make a program that implements the tilt angle protection of the motor controller. The functionality of the tilt angle protection is as follows:

- The tilt angle protection system defines a warning level of the tilt angle and an error level of the tilt angle. Their values are defined by reg[20] and reg[21].

- If the tilt angle warning level (reg[20]) is 0, the tilt angle warning system is disabled. In that case, the green and red RGB leds are both turned off. For a non-zero vale of reg[20], the green error led turns on if the measured tilt angle of the MPU6050 board is less than reg[20]. Otherwise, the red RGB LED is turned on.

- If the tilt angle error level (reg[21]) is set to 0, the tilt angle error system is disabled. In that case, the buzzer is turned off. For a non-zero vale of reg[21], the buzzer is off if the measured tilt angle of the MPU6050 board is less than reg[21]. Otherwise, the buzzer is turned on at a frequency of 500 [Hz].

Make in a new main.py a script with the following code.

1. Define an object of the MPU6050 class.

2. Define an object of the HighlevelController class. Fill in None as argument of tcp.

3. Define a function TiltAngle() that calls the method tiltAngle() of the MCU6050 object and writes the measured accelerations and tilt angle in elements reg[3] up to reg[6] of the HighLevelController object (see Appendix E). Then it compares the measured tilt angle to the tilt angle warning and error levels (reg[20] and reg[21]) and turns on and off the red and green RGB leds and buzzer as specified above.

4. Call in an endless loop of exactly 100ms the function TiltAngle() and method readFromUart() of the HighLevelController object.

Use KiTTY to test the program. Enter in KiTTY write request(s) to set the tilt angle warning level and tilt angle error level to some non-zero values. Tilt the ESP32 board to check if the buzzer and RGB leds turn on. Enter in KiTTY read requests to check the buzzer and RGB leds turn on at the correct tilt angle.

Add the code of main.py to your answer document.

4.9 Testing the WiFi communication

The ESP23 has a WiFi/Bluetooth module that we use to enable communication with a high level controller via a TCP-socket. A TCP-socket is the building block of most of the internet communication. The link: Socket Programming in Python (Guide) – Real Python gives a good description of how data exchange with sockets works and which steps are necessary to create a socket connection in Python. Read the start of it until the section 'Echo Client and Server'.

In our application, the motor controller acts as a server. It provides services such as making the motor run and providing sensor data to the client (the high level controller). In our application, the motor controller can only accept one client. It is not logic that multiple high level controllers control one motor. On the other hand, it is possible that the high level controller can be connected to multiple motor controllers (see figure 4.7). Most robots have more than one motor which are all controlled by one high level controller.

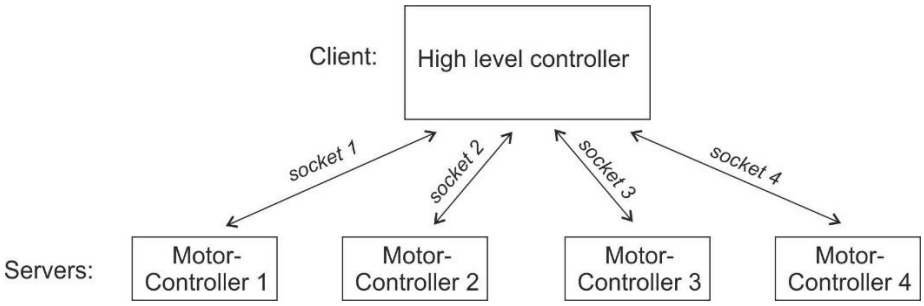


figure 4.7: Client – servers connections between one high level controller and 4 motor controllers.

To handle the connecting to a TCP-socket and the communication with it, we use on object of the class TCPHandler. The class defines the following constructor and methods for data exchange with a Client:

- TCPHandler(ssid, password, port, led=None). The constructor has 4 arguments:

- the name (ssid) and password (password) of the network to which it connects (e.g. a modem or router)

- the port number of the TCP connection

- A GPIO pin (led) which is set high if a connection with a client is made and low if the client is disconnected.

The constructor initiates a state machine that handles the connection to the network server and the connecting of a Client via a TCP-socket. Other useful methods of the TCPHandler class are:

- transmit(s): The method transmit(s) transmits string argument s via the socket if a connection is present.

- receive(): The return value of the method receive() is a bytes variable that contains all characters received from the socket since the last read or None if nothing is received.

- getState (). This method returns one of the following values:
  0: The ESP32 is not connected to a network
  1: The ESP32 is connected to a network but not connected to a client
  2: The ESP32 is connected to a client.

The constructor and methods transmit() and receive() are non-blocking. This means that the program doesn't halt until a connection is made with a client or data is available. For our application, this is necessary because the motor controller must also work if only a serial port connection is used for interfacing with a high level controller and no TCP-connection is used.

Question 23 (4 pnts)

Make in a new main.py that creates an object of the TCPHandler class. Enter Pin(2) as Pin argument led. Then the blue LED on the ESP32 board turns on if a client connection is make.

Read in an endless loop that is executed each 100 [ms] data that has been received via the TCP-socket. If data has been received, the program evaluates the data received. If it is string that represents a valid numerical value, it transmits a string to the client that holds the square of that number. If it is not a valid numerical value, it returns the string 'No valid number' to the Client. Figure 4.8 shows the flow chart of the program.

Add the code of main.py to your answer document.

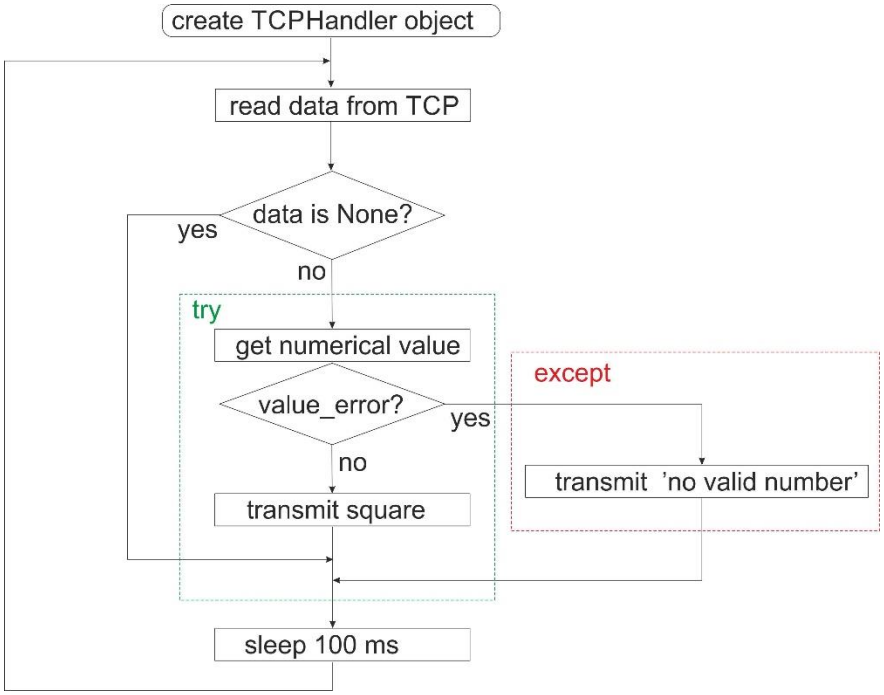
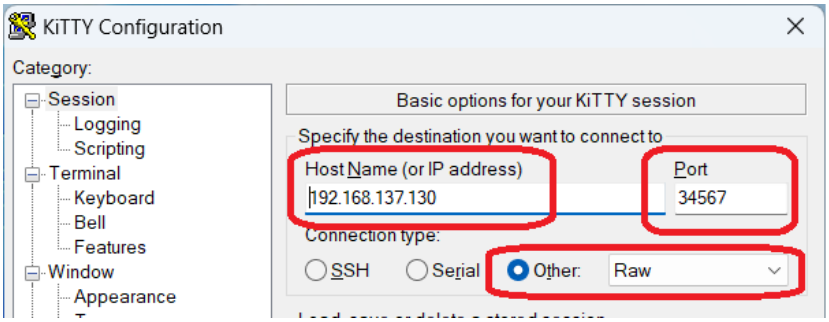


figure 4.8: flow chart of main.py.

Remarks:
- It is most easy to use the mobile hotspot on your laptop for the connection with the ESP32. Select the 2.4GHz band.
- The port of a TCP-socket is a 16-bit number higher than 1023 (port numbers below 1024 are reserved for standard applications such as http, mail, file transfer, etc.).
- Use the function eval(s) to evaluate a string to a number. Use a try block because the function eval() raises an exception (error) if its argument doesn't represent a valid number. Include the transmission of the square of the number inside the try block. Place the transmission of the string 'No valid number' in the exception block.
- Use KiTTY to test the program. Select at communication type: Other → Raw.
Enter the IP-address of the ESP32 and the port number used for the communication.



5 Integration tests

In the previous chapter, all subsystems of the motor controller have been tested. In this chapter, we will integrate all firmware and test the complete system. This is done in 3 steps:

First, you have to write a basic test program that uses all subsystems of the motor controller. If that works, you have to extend the program with a speed control / torque limit control. And finally works, you have to extend the program with a manual control.

The functionality of the last question worked out must be reviewed to assess the proper functioning of the program. You must make an appointment with the supervisor to do this.

Question 24 (14 pnts)

Write in a new main.py a program to test all functionality of the motor controller. The following requirements apply.

- The variables of the register space that relate to the motor characteristics (reg[13] up to reg[19], see appendix E) are initialized. Use the current measured in question 18 at half the rated speed for the no-load current. Use the motor data sheets for all other motor characteristics.

- The sensor data defined in reg[0] up to reg [8] can be requested by the TCP client. The speed of the shaft (reg[7]) must be calculated from the encoder frequency, gear ratio of the transmission and the number of pulses per revolution of the encoder. The torque at the shaft (reg[8]) must be determined from the motor current, rated torque and rated current (see appendix F).

- The pwm value of the half-bridges is set to the pwm1 values given by reg[22] and reg[23]

- The tilt angle protection, as described in question 22, is implemented.

Follow the steps below.

Initialize the register variables reg[13] up to reg[19], initialize the Encoder module and create objects of the classes MPU6050, TCPHandler and HighLevelController at the start of the program.

Execute the following in an endless loop that is carried out each 100ms.

- Read the battery voltage and the motor current and store their values in reg[0] and reg[1].

- Read the encoder frequency and store its value in reg[2]

- Call the function TiltAngle() of question 22. Extend the function with enabling or disabling the H-bridges in case of a tilt angle error.

- Calculate the speed and torque of the shaft as explained in appendix F and store the values in reg[7] and reg[8].

- Call the method readFromTCP() to exchange data with the TCP-client.

- Set the duty-cycle of the pwm pins driving the half-bridges equal to the values of reg[22] and reg[23].

Download and test the program with the app “MotorController.exe” (zipped in the file Motorcontroller.zip on Brightspace).

Add the code of main.py to your answer document.

If this is your final elaborated question, you must have it reviewed by the supervisor.

Question 25 (10 pnts)

In this assignment, you have extend the program of question 24 to a motor controller that implements 3 operation modes defined by the value of reg[9].

- Mode 0: The motor controller is turned off (the duty cycles of the half bridges are 0).
- Mode 1: Test mode. The duty cycles of the half bridges equal the contents of reg[22] and reg[23] (this operation mode is worked out in question 24).
- Mode 2: Speed control/torque limit mode. The duty cycles of the half bridges are controlled by an integrative controller to control the speed to the setpoint given by reg[10] and limit the motor torque to the value given in reg[11]. Appendix F describes how to work this out.

Follow the steps below.

- The maximum speed of the motor at 12V is about 250 [rpm]. Calculate the integrator constant k_i so that an error of 250 [rpm] (motor is standing still, setpoint is 250 [rpm]) leads to an increase of the duty-cycle from 0% to 100% in 0.25[s].

- Write a program that implements the flow-chart of figure 5.1.

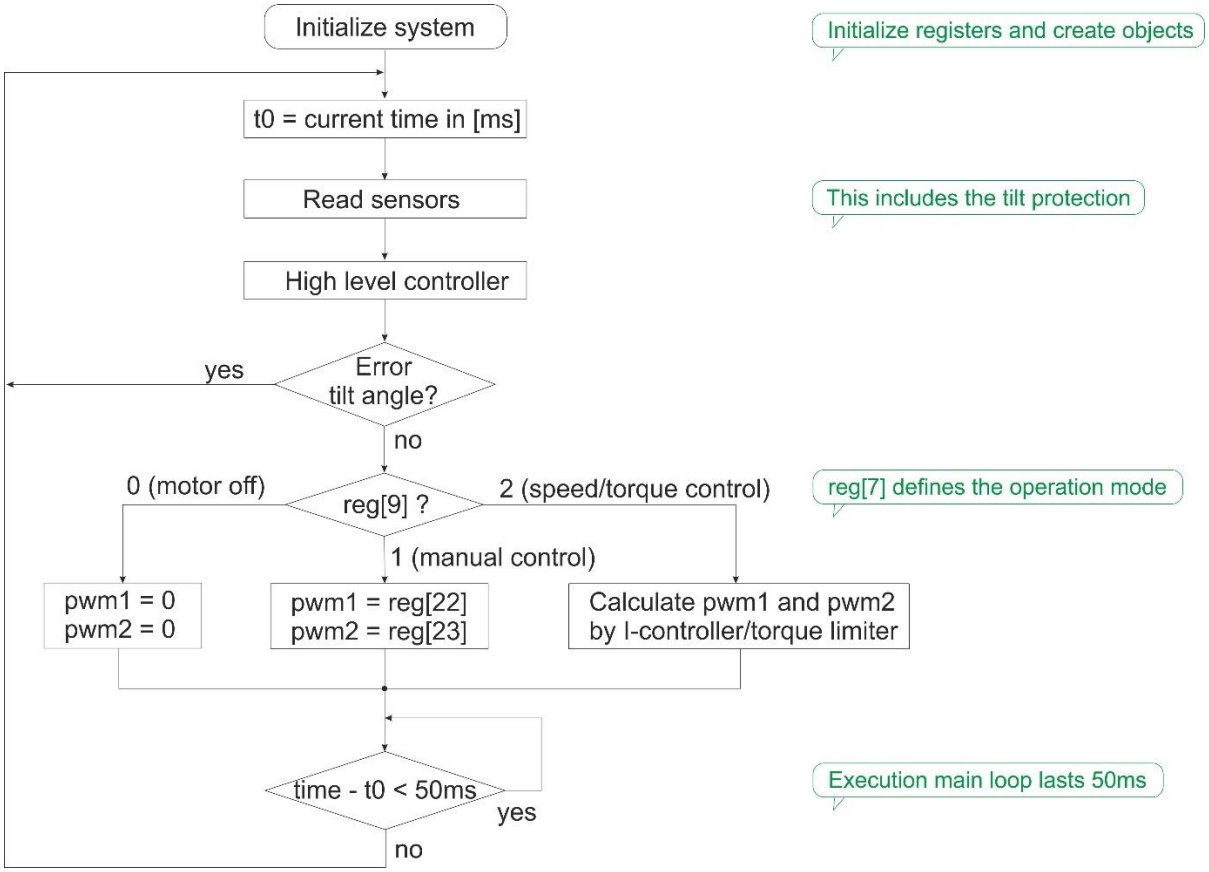


figure 5.1: flow-chart of motor controller application.

- Test operation mode 2 with positive and negative values of speed setpoints and torque limits. Fill in the 2 tables on the next page.

Table 24.1: Test results of the speed controller. Perform the measurements with an unloaded (free running) motor. Adjust the torque limit to the rated motor torque.

Speed setpoint [rpm]	Speed shaft [rpm]	Encoder frequency [Hz]	Difference speed setpoint and speed shaft [%]
25			
50			
100			
150			
200			
250			
-25			
-50			
-100			
-150			
-200			
-250			

Table 24.2: Test results of the torque limiter test. Perform the measurements as follows.
Mount a thickener on the shaft that allows you to manually block the shaft. Adjust the speed setpoint to 50 [rpm] and the torque limit to 50 [mNm]. Then, gently load the motor by snapping your hand on the thickener until the shaft speed decreases. Write the motor current and torque in the 2nd and 3rd row of the table below. Repeat this procedure for torque limits of 100, 150 and 200 [mNm].
Carry out all measurements again for a negative speed setpoint of -50 rpm.
Note: The motor current must not exceed 1.5 [A]. If it does, something is wrong and you must stop the measurement before the motor or motor controller gets damaged.

Torque limit [mNm]	Torque at shaft [mNm]	Motor current [mA]	Difference torque limit and measured torque [%]
Speed setpoint= 50 [rpm]			
50			
100			
150			
200			
Speed setpoint= -50 [rpm]			
50			
100			
150			
200			

Add the tables filled in plus the code to your answer document.

If this is your final elaborated question, you must have it reviewed by the supervisor.

Question 26 (10 pnts)

In this assignment, you have to extend the program of question 25 by implementing a user interface of the motor controller that works as follows.

- The user can alternately turn on the motor by pressing pushbutton 1. The following holds.
  - Operation mode 1: A button press turns off the motor by setting the control mode (reg[9]) to 0
  - Operation mode 2: A button press turns off the motor by setting the control mode to 0
  - Operation mode 0: A button press will set the control mode back to the control mode that was active before operation mode 0 was selected.

  Nota bene: if there is a tilt angle error, the motor is always turned off.
- The user can increase the motor speed by pressing push button 2. This works as follows.
  - Operation mode 1:

    If pwm2 (reg[23]) is 0, then pwm1 (reg[22]) is increased by 50 with each button press until it reaches a maximum value of 970.

    If pwm2 is not 0, pwm2 is decreased by 50 with each press until it is 0.
  - Operation mode 2:

    The speed setpoint (reg[10]) is increased by 5% of rated speed with each press until the speed setpoint has reached a maximum value of 120% of rated speed.
- The user can decrease the motor speed by pressing push button 3. This works as follows.
  - Operation mode 1:

    If pwm1 is 0, then pwm2 is increased by 50 with each button press until it reaches a maximum value of 970.

    If pwm1 is not 0, pwm1 is decreased by 50 with each press until it is 0.
  - Operation mode 2:

    The speed setpoint (reg[10]) is decreased by 5% of rated speed with each press until the speed setpoint has reached a minimum value of -120% of rated speed
- Any change of a register caused by the user is notified to the high-level-controller. When the users presses pushbutton 1 to enable or disable the H-bridge, the program sends a read response of address 9. If the speed is changed by pressing pushbutton 2 or 3, the program sends a read response of the register that is changed (address 10, 22 or 23)). Use the method transmitReadResponse() of the HighLevelControl class for sending read responses.

  Note: don't send read responses in an interrupt. This can interfere with the processing of read requests in the main loop of the program. If you use interrupts to detect a button press, it is better to set a flag and then process the flag setting in the main loop.
- Initialize the control mode (reg[9]) to a value of 2 and the torque limit (reg[11]) to the rated torque of the motor. This makes it possible for a user to work with the motor controller without a high-level controller.

Add the code of main.py to your answer document.

If this is your final elaborated question, you must have it reviewed by the supervisor.

Appendix A Bill of materials

Name	Designator	Footprint	part*
10kΩ	R23,R37,R39,R40,R41,R42,R43	R0805	C17414
470uF	C19	CAP-TH_BD10.0-P5.00	C88740
PMEG6020ELRX	D4,D3,D1,D2	SOD-123	C478000
PZ254V-11-01P	JCS+,J5V,JGND	HDR-TH_1P-P2.54-V-M	C492400
TF2104M-TAH	U1,U2	SOIC-8	C2683754
33uH	L1	IND-SMD_L7.0-W6.6	C177245
330Ω	R25,R22,R24,R26,R27,R28,R29	R0805	C17630
4.7kΩ	R33,R34,R35,R36	R0805	C17673
SEA-1285F-42Ω-40P6.5	BUZZER	BUZ-TH_BD12.0-P6.50	C2693578
22uF	C1,C2,C3,C4,C5,C6,C7,C8,C9,C10	C1206	C12891
330nF	C12,C13,C14,C15,C16,C17,C18,C20,C21	C0805	C1740
100nF	C22,C23,C24	C0805	C28233
22nF	C25	C0805	C1729
150Ω	R13,R14,R15,R16,R17,R18,R9,R10,R11,R12	R0805	C17471
220Ω	R19,R20,R21	R0805	C17557
470Ω	R30	R0805	C17710
1kΩ	R31	R0805	C17513
100kΩ	R45	R0805	C149504
PM254-1-15-Z-8.5	J1_ESP32,J2_ESP32	HDR-TH_15P-P2.54-V-F	C2897378
FC-2012HRK-620D	LED1,LED2	LED0805-RD_RED	C84256
D882_C9634	Q5	SOT-89-3	C9634
1Ω	R49	R0805	C25271
10Ω	R5,R6,R7,R8	R0805_NEW	C17415
TC5050RGBF08-3CJH-AF53A	RGB2	LED-SMD_6P	C784540
TS-1187A-B-A-B	SW1,SW2,SW3	SW-SMD_4P	C318884
MCP6001T-I/OT	U3,U4	SOT-23-5	C116490
INA181A1IDBVR	U5	SOT-23-6	C2058943
LMR14050SDDAR	U6	SOIC-8	C155484
Header-Male-2.54_1x3	A0,POWER,UART	HDR-TH_3P-P2.54-V-M-1	C49257
Header-Male-2.54_1x6	MOTOR	HDR-TH_6P-P2.54-V-M-1	C37208
MPU6050	MPU6050	HDR-TH_8P-P2.54-V-F	C2685213

* part code is of LSLC (Electronic Components Distributor - LCSC Electronics)

** The components C11, R,1 R32, R38, R44 and Q1 up to Q4 are not given. You have to find them yourself.

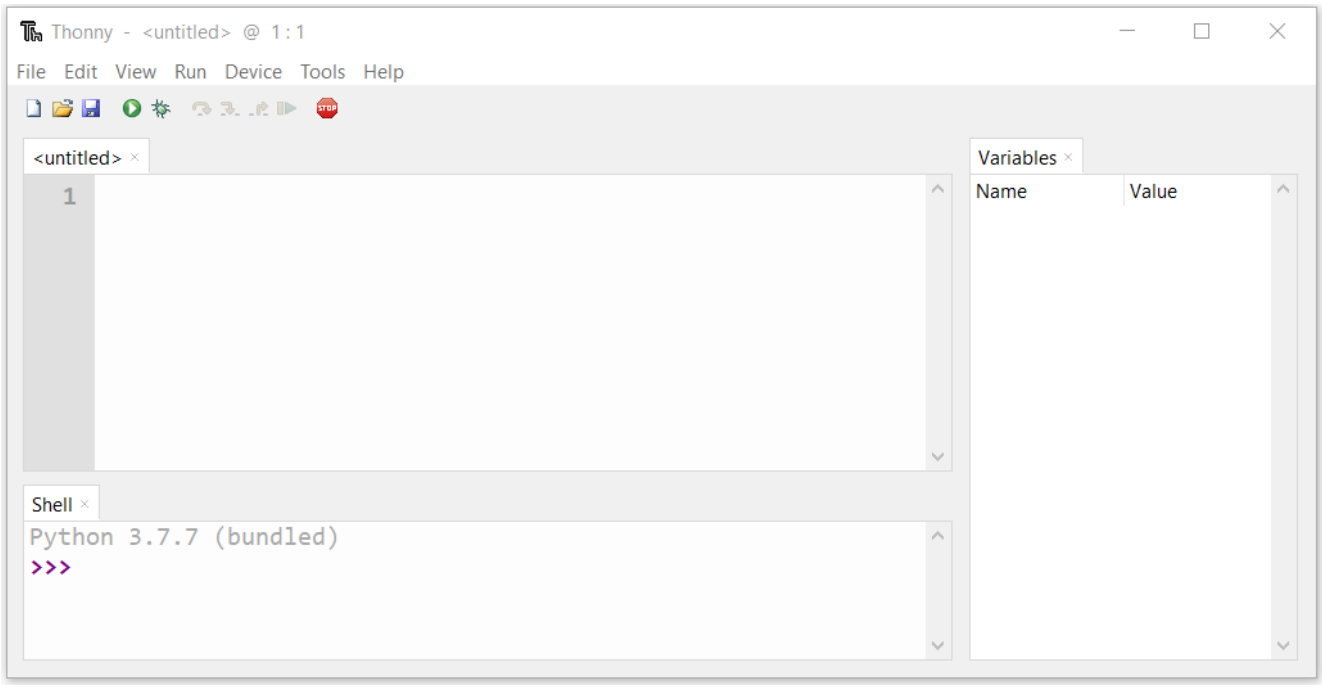
Appendix B Software development environment ESP32.

In this course, we use Thonny as IDE. Thonny is a light-weighted but nevertheless a versatile software development environment. It is particularly suitable for this course due to its simplicity. It includes the following options, among others:

- The Python interpreter
- A code editor with syntax highlighting
- A Python command window
- A variable window
- A debugger (allows the programmer to debug)
- A run button that allows you to run a script file directly by the Python interpreter
- A bundle with several commonly used libraries
- A library installation tool

Download and install Thonny (https://thonny.org/).

Start Thonny. When you open it for the first time, you probably see 2 windows. The top window is a file window. Here you can open and edit python scripts. The bottom window is the Python command shell. Here you can input Python commands directly. In general, it is also handy to make the variables window visible (View → Variables).



To test Thonny, type (or copy-paste) the Python code that is given below in the top window, save it and run it by pressing the run-icon ( )or by pressing the <F5>- key. Check it the code works as you expect.

```
from random import random          # Import the random function
while True:
    name = input('Input your name (press only enter key to stop): ')
    if name == '': break
    x = random()
    if x<0.5:  print(name, 'is a stupid name')
    else:      print('Welcome', name)
print('End of program')
```

To run python scripts on the ESP32 board first you have to download and flash a MicroPython binary image on the ESP32. This is done as follows:

Step 1: Go to [MicroPython - Python for microcontrollers] and download the latest stable firmware.

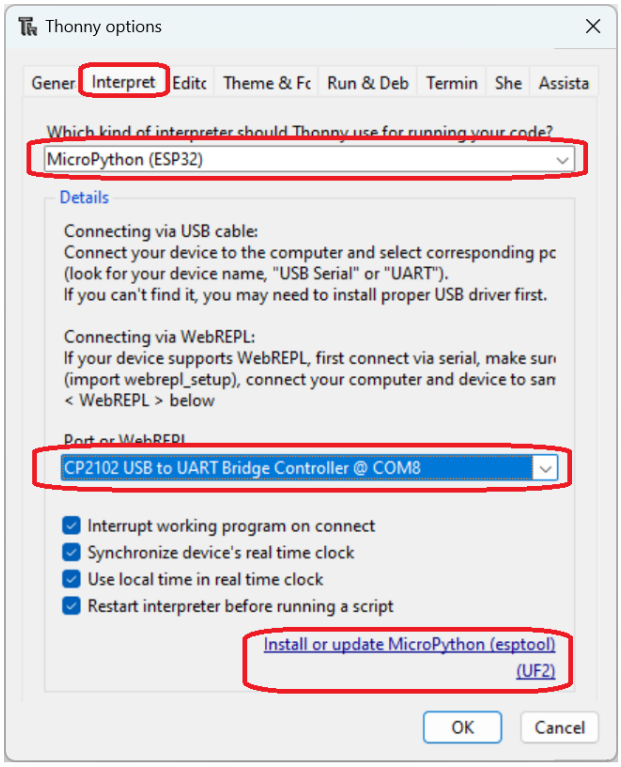
Firmware

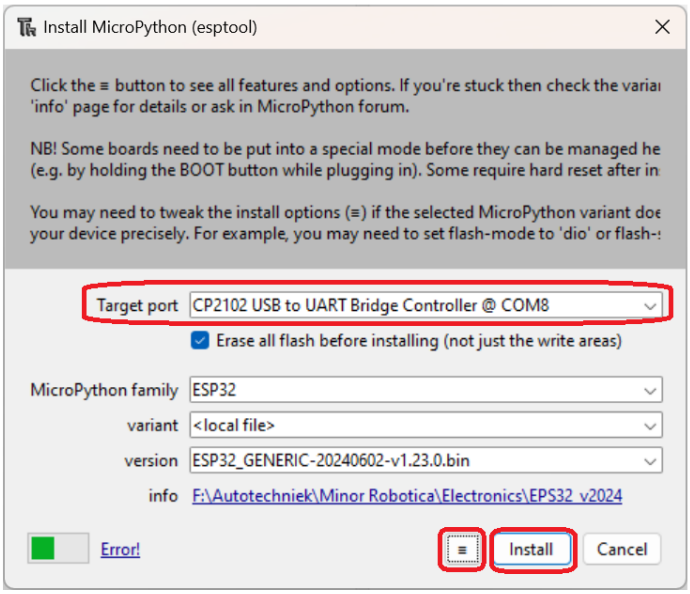
Releases

v1.26.0 (2025-08-09) .bin / [.app-bin] / [.elf] / [.map] / [Release notes] (latest)
v1.25.0 (2025-04-15) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.24.1 (2024-11-29) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.24.0 (2024-10-25) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]

Step 2: Connect the ESP32 board to your computer and select in Thonny the MicroPython interpreter via:

Tools → Options → Interpreter → Select MycroPython (ESP32) in listbox.





Select the CP12-2 USB to UART Bridge controller and click on the Install or update MicroPython (esptool) link. Check [CP210x USB to UART Bridge VCP Drivers - Silicon Labs (silabs.com)] for the latest drivers if it doesn't appear.

In the window that appears, enter again the CP12-2 USB to UART Bridge controller. Click on the button with the three dashes in the lower part of the window and select the binary image file that you downloaded in step 1 at Select local MicroPython image.

Click Install. The installation takes about 1 minute.

Remark: Press the En-button on the ESP-board if the flashing doesn't start

To run a python script or commands from the command shell on the ESP board, you have to ensure that the MicroPython (ESP32) interpreter in Thonny is selected (normally, this is only necessary if you have changed the interpreter settings).

Check whether the installation has been successful by typing the following commands in the command shell:

```
>>> import machine, esp
>>> f = machine.freq()     # variable f holds the CPU frequency (160 MHz)
>>> s = esp.flash_size()   # variable s holds the flash size of the ESP32 (≈4.2 MB)
```

If you have the ESP32 board working, we can make our fist program. It is a very simple program that makes the blue LED on the ESP32 board flash with a frequency of 1 [Hz]. The blue LED is connected to GPIO2. The application also sends the time that the application runs to the command shell of Thonny.

A MicroPython program is built around 2 files. The first file is called boot.py. This file is executed directly after powering the device or a hard reset. The file is located in the flash memory of the ESP32. You can see its contents by opening it: File → Open → MicroPython device → boot.py.
Note that the file name [boot.py] is placed in between square brackets on the tab. The square brackets indicate that this file is located in the flash memory of the ESP32. After flashing the MicroPython image, boot.py is basically an empty file with only comments in it (see figure below).

[boot.py]

```
1  # This file is executed on every boot (including wake-boot from deepsleep)
2  #import esp
3  #esp.osdebug(None)
4  #import webrepl
5  #webrepl.start()
6
```

Normally, boot.py contains code to initialize the ESP32. For this application, we only have to define the GPIO pin to which the LED is connected as an output pin. This can be done by the code shown in the figure below.

[boot.py] ×

```
1  import esp
2  esp.osdebug(None)
3
4  from machine import Pin
5  LED = Pin(2, Pin.OUT)
```

The code on lines 1 and 2 is standard code. This must be left unchanged for each application. In line 4, the Pin class of the module machine is imported. It is used in line 5 to create an object LED that represents GPIO 2 and it configures this pin as an output pin. Enter this code given and save it in the flash memory of the ESP32. Save it also on your computer, so you have a backup if something goes wrong. You can save it on your computer via:

File → Save As → This computer → Select the folder where you want to save it.

Remark: Be aware that you may have 2 versions of a python script: one on your computer and one on your ESP32. To avoid confusion, check which version you are working on (this can be seen if the filename is placed within square brackets or not).

Press the reset button on your EPS32 board (the button labeled EN). If everything works, then the Variables window shows among other the object LED.

Check if it the LED turns on by turning the command below in the command shell:

```
>>> LED.value(1)
```

To turn on the LED, type in the command shell:

```
>>> LED.value(0)
```

The code to make the LED flash and to send the running time of the app to the command shell is placed in a script file called main.py. This script file is executed after boot.py and it contains normally the functionality of the application. Create a new script file with the contents shown below and save it on the ESP32 and on your computer. Then press the reset button of the ESP32 board and see if it works.

[main.py] *

```
1  import time                  # This module has timing related functions
2
3  t = 0
4  while True:
5      LED.value(1)             # turn on LED
6      time.sleep_ms(500)       # sleep for 500 milliseconds
7      LED.value(0)             # turn off LED
8      time.sleep_ms(500)       # sleep for 500 milliseconds
9      t = t + 1                # increase time
10     print('Time elapsed:', t, '[s]')  # print time
```

Remarks:

1. A hard reset is given by pressing the reset button on the ESP board. This resets the Python interpreter and then it runs boot.py and main.py. If a program hangs completely, this is the best way to reset it.

2. A soft reset is given by pressing the <Ctrl>+D key combination in Thonny. It stops the execution of the current program and then it runs only main.py

3. A running program can be interrupted by pressing the <Ctrl>-C key combination. This stops the execution of the current program, it does not start main.py.

4. By pressing the run-icon () or <F5>- key, you run the active script in the file-window. This should be done only if no other script is running already. If it is, stop it via <CTRL>+C.

For this course, you need the following 4 python scripts that can be found in a zip-file on Brightspace.

Encoder.py:	This script contains code to read the encoder frequency
MPU6050_studentversion.py.	This script contains code to communicate with the mpu6050.
TCPHandler.py:	This script contains code to communicate via a TCP socket
HighLevelController.py:	This script contains code to communicate with a high level controller.

Download and unzip PythonScripts.zip and flash all 4 script files on the ESP32 so that you can use them when you need them.

Appendix C Reading the encoder frequency

The most accurate way for microcontrollers to determine the timing related phenomena, such as a frequency, is by using a capture module. The ESP32 capture module consists of a counter that is reset at a gpio pin change event and a capture register that can read the counter output at a gpio pin change event. The ESP32 has 2 timer/capture modules that both run on a 80MHz-clock signal (see figure C1).

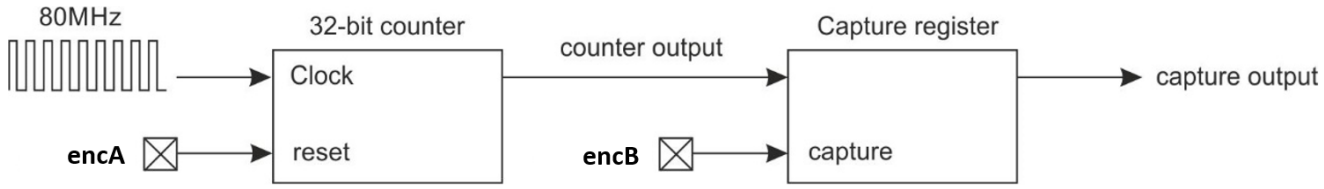


Figure C1. Block diagram of a timer/capture module

To read the encoder frequency, we use both timer/capture modules of the ESP32. One timer/capture module measures the time interval from a rising edge at encoder A (encA) input to the falling edge of encoder B (encB) input. The other timer/capture module measures the time interval from the falling edge of encB to the rising edge of encA. The frequency of the encoder is then determined by the sum of both time intervals, the speed direction can be determined by checking which of both intervals is the largest.

The python script file Encoder.py defines 2 user functions to be used for reading the frequency of an encoder.

The function InitEncoder(gpio1, gpio2) is used to configure the 2 capture/timer modules. Normally, this function should be called only in the initialization part of the program. Arguments gpio1 and gpio2 are the pin numbers to which the encoder signals encA and encB are connected.

The function GetFrequency() is used to read the frequency of the encoder.

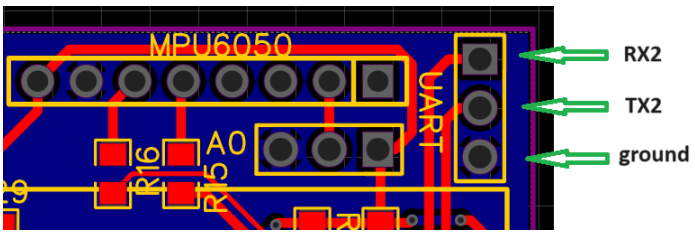
The file Encoder.py should be flashed on the ESP32 (see Appendix B).

Appendix D USB-serial port adapter and terminal emulator

For communication with the serial port, we will use a CH340 or Profilic USB-serial or a port adapter. The USB-serial port adapter has one side 4 pins, that must be connected to the serial port header UART on the motor controller PCB (see description below.

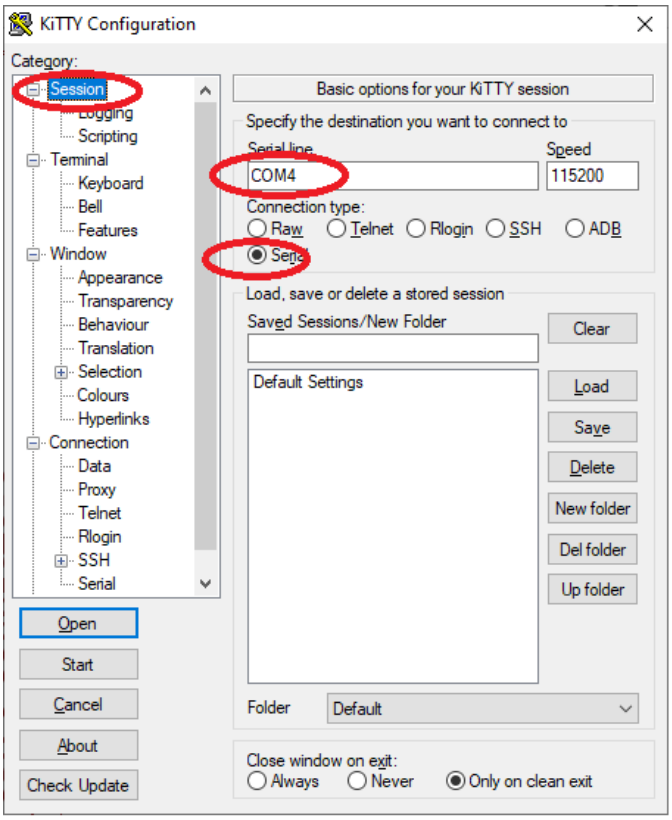
Profilic USB-serial	CH340 USB-serial	UART connector on PCB
TXD (green wire)	TXD	RX2
RXD (white wire)	RXD	TX2
Black wire (ground)	Ground	Ground pin

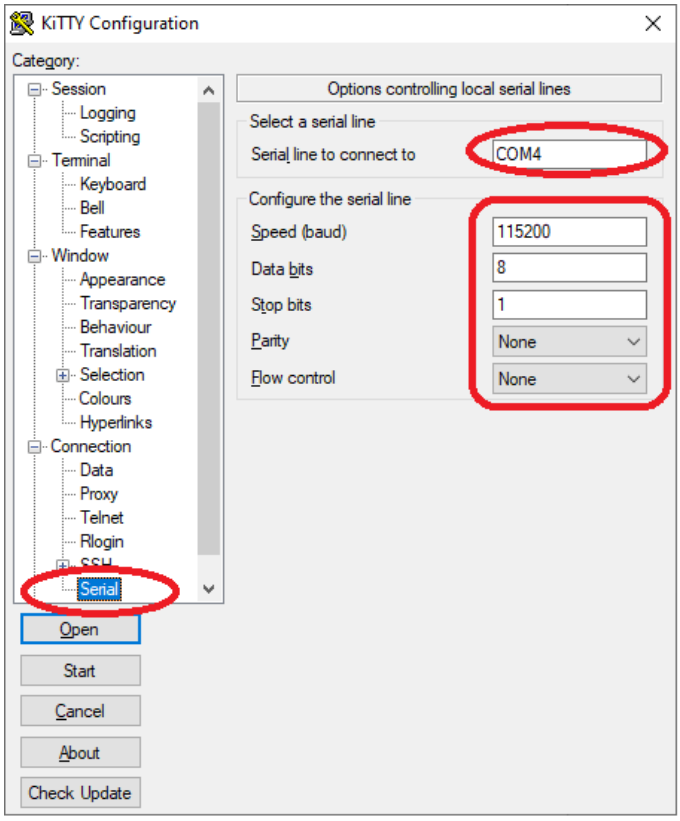
*Remark: The Profilic adapter is easier to use because it is already wired. However, Windows 11 doesn't support the profilic usb-serial port adapter. Follow the instructions of Installation PL2303TA for windows 11.pdf (*Brightspace*) to get it work. If it doesn't work, use the CH340 adapter.*

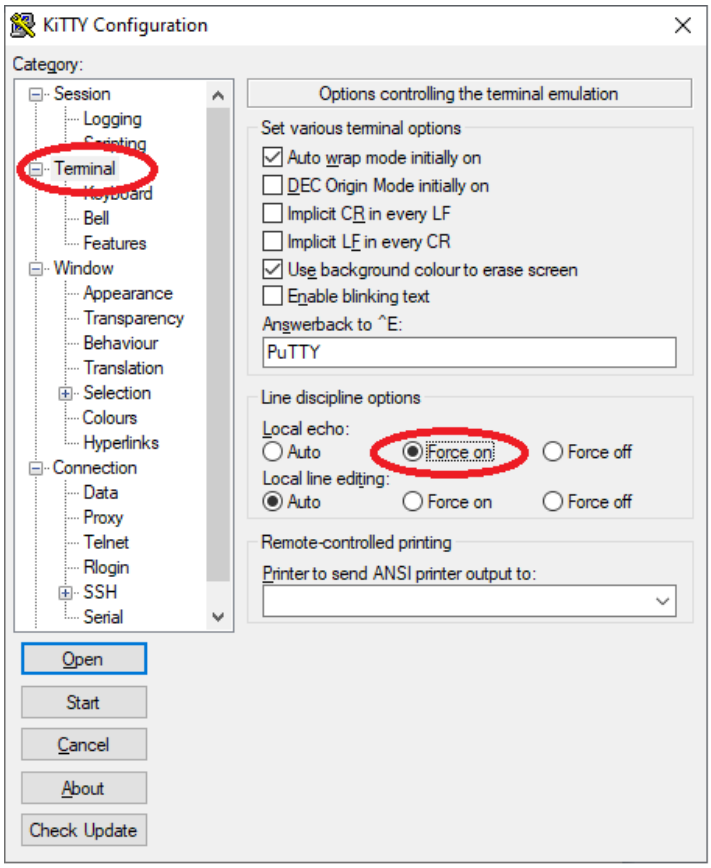


We use Kitty as terminal emulator program. A terminal emulator is a program that is used to transmit and receive text messages via the serial port. You can download KiTTY from: [Download KiTTY latest release (fosshub.com)]. Before adjusting the right settings in KiTTY, you have to find out which COM-port is assigned to the USB-to-serial adapter. You can do this in device manager (Dutch: 'Apparaatbeheer') in Windows. Look for the Profilic or CH340 USB-to-Serial Port in the category Ports (COM & LPT).

Run Kitty and fill in the settings below (replace COM4 by the COM port number found in the device manager).







Appendix E Communication protocol

The motor controller defines a 64 variables wide register space for exchanging data with a high level controller. This register space can be accessed via the serial port or via a TCP-socket. All variables are 32-bits wide integers. The address range is from 0 up to 63 (see figure F1).

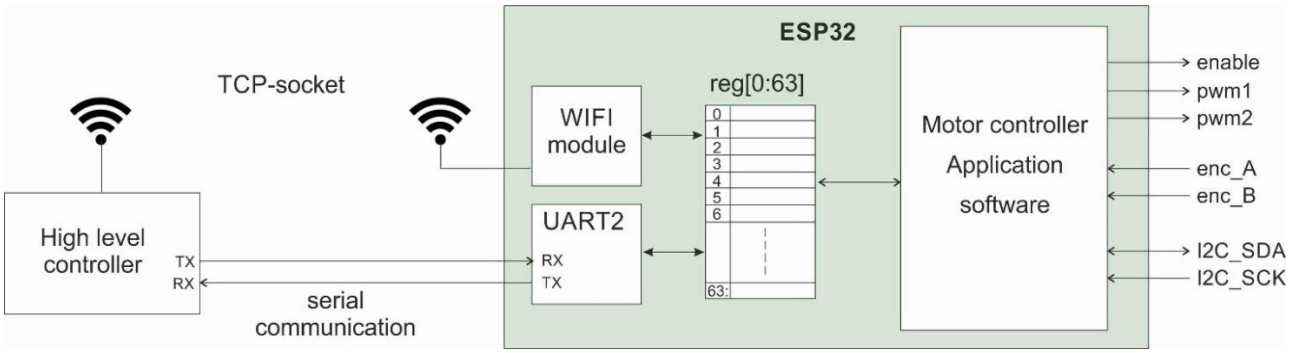


figure F1: Communication interface structure of the motor controller.

The table below shows the contents of the register space. The word shaft refers to the output shaft.

Address	Signal	Unit	Remark
0	Battery voltage	[mV]	Sensor data
1	Motor current	[mA]	Sensor data
2	Encoder frequency	[Hz]	Sensor data
3	Acceleration in x-direction	[mm/s^2]	Sensor data
4	Acceleration in y-direction	[mm/s^2]	Sensor data
5	Acceleration in z-direction	[mm/s^2]	Sensor data
6	Tilt angle	[°]	Sensor data
7	Speed shaft	[rpm]	Sensor data
8	Torque shaft	[mNm]	Sensor data
9	Control mode	-	0: H-bridges disabled 1: Test mode 2: Speed control/Torque limit
10	Speed setpoint	[rpm]	Setpoint of shaft speed in control mode 2. The sign of the speed setpoint determines the direction of rotation.
11	Absolute value torque limit	[mNm]	Torque limit at shaft in control mode 2. Must be positive.
12	Integrator controller constant	[10^{-3} %/s/rpm]	Used in control mode 2
13	Encoder pulses per revolution	[-]	Encoder characteristic
14	Gear ratio transmission	[10^{-3} rpm/rpm]	Transmission characteristic
15	Rated speed	[rpm]	Speed of shaft
16	Rated torque	[mNm]	Torque of shaft
17	Rated current	[mA]	Rated motor current
18	No load current	[mA]	No load current motor
19	Armature resistance	[mΩ]	
20	Tilt angle warning level	[°]	A zero value disables the tilt angle warning protection
21	Tilt angle error level	[°]	A zero value disables the tilt angle error protection
22	pwm1 value test mode	[-]	Range: 0 ≤ pwm1 ≤ 980
23	pwm2 value test mode	[-]	Range: 0 ≤ pwm2 ≤ 980
25-63	Reserved for future use		

The register space can be accessed by a high level controller via ASCII strings (see: ASCII (tekenset) - Wikipedia).

- Reading data from the register takes place in 2 steps:

Step 1: The high level controller transmits a read request. A read request consists of an address followed by the number of variables to be read in between brackets:

(a,n) where: a = address of first variable to be read

n = number of variables to be read

Step 2: The motor controller responds with a read response. A read response has the following structure:

$(a,x_a,x_{a+1},x_{a+2},\ldots, x_{a+n-1})$ where: a = address of first variable

x_a = value of the variable at address a

- Writing data takes place in one step by transmitting a write command. A write command consists of an address followed by an integer within square brackets:

$[a,x_a]$ where: a = address of variable

x_a = value of the variable written to address a

Note 1: spaces and tabs are not allowed inside the brackets.

Note 2: the character sequence '_q&' is used as an escape sequence in the TCP communication. If the motor controller receives '_q&', it disconnects the high level controller.

Appendix F Estimation of the torque at the shaft

The motor controller doesn't have a sensor that measures the torque at the shaft directly. Therefore, we will use the motor current sensor to estimate the shaft torque. To do so, we assume that the transmission has a constant efficiency. In that case, the following holds:

$T_{motor} = k_T \cdot I_{motor}$ where: k_T = torque constant motor

$T_{shaft} = (T_{motor} - T_{friction}) \cdot i \cdot \eta$ where: i = gear ratio of transmission, η = efficiency of transmission

We can work out these equation further as:

$$T_{shaft} = \eta \cdot i \cdot \left(k_T \cdot I_{motor} - T_{friction}\right) = \eta \cdot i \cdot k_T \cdot \left(I_{motor} - \frac{T_{friction}}{k_T}\right) = k_{TMT} \cdot (I_{motor} - I_0)$$

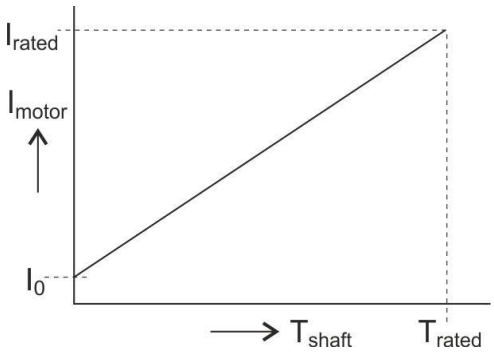
where: k_{TMT} = torque constant of motor combined with transmission, I_0 = no-load current

In general, the friction torque and no-load current depend on the motor speed. For the motor that we use, the no-load current varies between about 100mA (at low speed) and 200mA (at rated speed). In the software, we do not take the speed dependency of I_0 into account. In order to get an acceptable accuracy over the whole speed range, it is best to use the no-load current at half of the rated speed. In that case, the calculated torque at low speed is a little too low or it can even be negative if the motor is not loaded. At high speed, the calculated torque is a little too high.

Remarks:

1. In practice, many datasheets of motors don't give the torque constant of the motor and/or the efficiency of the transmission. In that case, the constant k_{TMT} must be calculated from the torque-current graph or rated torque and current (see figure on the right).

Note that the register space of the motor controller only defines variables for the rated torque and current (reg[15]) and reg(16]). So, in the software, you must use them to calculate k_{TMT} and the torque on the shaft.



2. In data sheets, the sign of the no-load current is almost always positive. But in real life, the sign of no-load depends on the direction of rotation. When the motor is driving in the negative direction, the no-load current I_0 should be taken negative in the calculation of the shaft torque.

Appendix G Speed controller with torque limiter

Figure F1 show a block diagram of the speed controller with torque limiter. The integrator block is the main part of the speed control; the limiter block provides torque limiting.

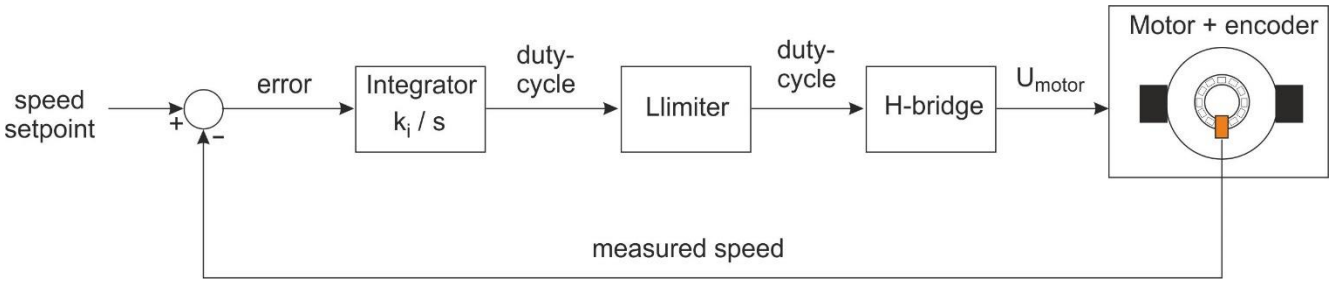


figure F1: speed control system with torque limiter

The speed controller works as follows. First, the error is determined as the difference between the desired speed (setpoint) and the measured speed. Next, the error is multiplied with a constant k_i and integrated. The output of the integrator provides the duty-cycle that goes via the Limiter block to the H-bridge.
If the motor runs too slow, the error is positive. At a positive error, the integrator output (the duty-cycle) will increase. This leads to an increasing motor voltage and the motor will accelerate.
If the motor runs too fast, the error is negative. Then, the output of the integrator decreases, so the duty-cycle and motor voltage decrease and the motor will decelerate.
If the motor runs at the desired speed, the error is zero. The output of the integrator remains the same and the duty-cycle and motor voltage will remain the same.

The constant k_i of the integral controller has a strong influence on the behavior of the system. If it is relatively small, the changing of the duty cycle goes relatively slow and it takes a relatively long time before the steady state situation arises. A relatively high k_i makes the changing of the duty-cycle go faster, leading to a more aggressive response. However, if k_i is too high, overshoot may occur or the system can get instable.

The rate of change of duty-cycle Δdc in a time interval Δt can be calculated as follows:

$$dc = \int k_i \cdot error \cdot dt \quad \rightarrow \quad \frac{\Delta dc}{\Delta t} = k_i \cdot error \quad \rightarrow \quad \Delta dc = k_i \cdot error \cdot \Delta t \qquad (dc = duty - cycle)$$

The code below shows how to implement an integrator controller in Python:

```
error = setpoint_speed - measured_speed   # calculate error
Delta_dc = ki * error * tsample           # Delta_dc = change duty-cycle
duty_cycle += Delta_dc                    # add change of duty-cycle to previous duty-cycle
```

In this code, tsample is the time between 2 successive executions of this code.

To add the torque limiter in the control, we must know the torque on the shaft. Since the PCB doesn't have a sensor that measures the torque of the shaft, we will use the motor current sensor to estimate the shaft torque. The following applies (see appendix F)

$T_{shaft} = k_{TMT} \cdot (I_{motor} - I_0)$ where: k_{TMT} = torque constant of motor + transmission and I_0 = no-load current

To limit the shaft torque, the motor current must be limited to.

$|T_{shaft}| < T_{max} \quad \rightarrow \quad |I_{motor}| \leq I_{max}$ where: $I_{max} = T_{max} / k_{TMT} + I_0$

In the speed control algorithm, described above, the duty-cycle changes at each execution step with Δdc. This change of the duty-cycle leads to a change in the motor voltage of: $\Delta U_{motor} = \Delta dc \cdot U_{in}$. Since the inertia of the system prevents the motor speed to change instantaneously, the change of motor voltage initially arises completely across the armature resistance. This gives a change in current of:

$\Delta I_{motor} = \Delta U_{motor} / R_a = \Delta dc \cdot U_{in} / R_a$

Assume the measured current is $I_{measured}$. Then, the expected current caused by the change in duty-cycle will be:
$I_{motor} = I_{measured} + \Delta I_{motor} = I_{measured} + \Delta dc \cdot U_{in} / R_a$

For positive speed setpoints, the motor current is positive and it holds:

$I_{motor} \leq I_{max} \rightarrow I_{measured} + \Delta dc \cdot U_{in} / R_a \leq I_{max} \rightarrow \Delta dc \leq R_a \cdot (I_{max} - I_{measured}) / U_{in}$

For negative speed setpoints, the motor current is negative and it applies:

$-I_{motor} \leq I_{max} \rightarrow -I_{measured} - \Delta dc \cdot U_{in} / R_a \leq I_{max} \rightarrow \Delta dc \geq -R_a \cdot (I_{max} + I_{measured}) / U_{in}$

The limiter block if figure F1 must limit the duty-cycle change to the equation given above. Also, the duty-cycle must be limited in a range between -0.98 ≤ duty-cycle ≤ 0.98 because outside this range the half-bridges don't work (see question 17b).

Remark:
To implement the I-controller in software, you can best define a separate function and name it for instance I_controller(). In this function, you have to use a <u>global</u> variable to store the duty-cycle calculated by the I-controller. You must make it global, because only then the value will be available at the next call of the function. Reset the variable when the I-controller is not active (if the user selects the manual control or motor off operation mode).