Contents

Listings

List of Figures

Chapter 5

Electronics Design

Summary

In this comprehensive guide, we will walk you through the step-by-step process of assembling the electronics for your exoskeleton and applying essential improvements. Consider this manual as an informative substitute for Chapter 5 in the EduExo Pro Handbook. It serves as a detailed reference to ensure a seamless and enhanced assembly experience.

5.1 The components - overview and functionality

The main electronic components of the *Classroom Exo* (Figure 1) are listed in , which were combined to an Open-Source Printed Circuit Board (PCB). The PCB design was completed in KiCad, the component files, part lists and production files can be accessed on our Github.

Table 5.1: *Component Overview*

Component	Functionality
Power Supply	5V@1.5A AC-DC converter
Battery Management Circuit	De-/Charging LiPo Battery Unit
Servo Motor	Actuating Elbow Joint
10kg Strain Gauge	sensing Forces at the wrist cuff
Surface Electromyography (sEMG)	Sensing sEMG signals at the Biceps and Triceps
Arduino Nano 33 IoT	Control of the *Classroom Exo* and communication with MATLAB GUI

5.1.1 Printed Circuit Board (PCB)

The heart of the *Classroom Exo* system is the Printed Circuit Board (PCB), shown in Figure 5.1. The Classroom Exos Printed Circuit Board (PCB) enhances the exoskeletons performance by upgrading to a 2-cell Li-Polymer Battery with a battery management Integrated Circuit (IC), ensuring optimal power for operation. With reachargeable batteries the replaceable batteries, maintenance is simplified. The streamlined connection process and safety features make *Classroom Exo* more user-friendly, especially for educators and students. Featured components include the BQ25886 battery management IC for efficient power regulation, Arduino Nano IoT 33 for control and communication, a Universal Serial Bus (USB) port for battery charging, a Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) switch for servo control, and 5 terminal blocks connecting the battery, Surface Electromyography (sEMG) sensors, servo motor, and force sensor to the PCB.

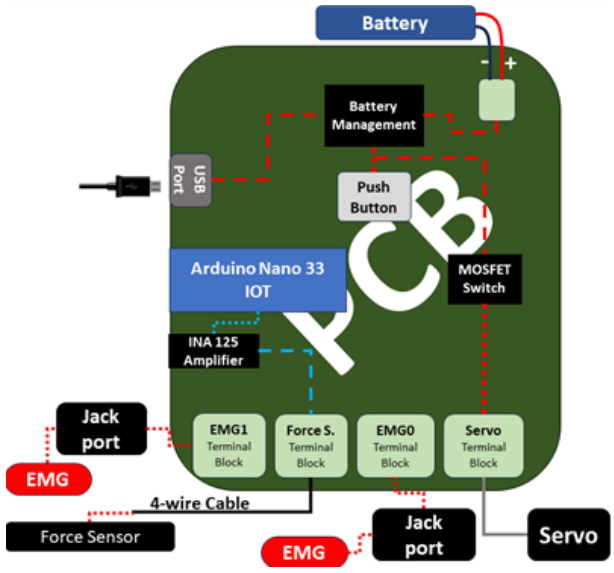


Figure 5.1: *The schematic illustrates the key components of the Classroom Exo PCB.*

To understand how our PCB works, let's talk about some important components first.

Resistors and Capacitors

Resistors control the flow of electricity in a circuit. They can limit the current and voltage or divide voltage in specific parts of the circuit. Imagine them as traffic controllers, directing the flow of electricity. They're also like volume knobs, adjusting the electric power in different areas. In circuits, resistors are often used to make sure devices like transistors work just right. They can even act as bodyguards, protecting components from too much electricity.

Capacitors, on the other hand, are like energy storage units in a circuit. They help smooth out bumps in electric power, making sure everything runs steadily. When there's a brief interruption in power, capacitors step in and provide a quick boost. They're also excellent at cleaning up messy signals. Think of them as noise-canceling headphones for electronic signals. In power supplies, capacitors are like stabilizers, keeping the voltage at a steady level. They're handy in amplifiers and other circuits, acting like traffic barriers to allow only certain signals to pass through.

Battery Management (BQ25886)

The BQ25886, integrated into the PCB design of the *Classroom Exo*, serves as a Battery Management IC with multifaceted functionalities. Primarily, it takes care of efficiently charging the 2-cell Li-Polymer battery and optimizing power delivery to the exoskeleton. The BQ25886 contributes to the longevity of the battery by controlling the charging process, preventing overcharging or undercharging. Moreover, the IC includes safety mechanisms, indicated by the integrated Light Emitting Diodes (LEDs) on the PCB.

The red LED signifies ongoing charging, low charge, or charge completion, while the blue LED indicates a reliable input source. In case of faults, the red LED blinks, alerting users to potential issues. More detailed information about the status LEDs can be found in the data sheet under section 8.3.8 Status Outputs. The BQ25886 acts as a robust guardian for both the microcontroller and the battery, ensuring the reliable and safe operation of the *Classroom Exo*.

Micro USB-Port & Power Supply

We use a 2-cell Li-Polymer battery that supplies a voltage ranging from 7.4 to 8.4 volts, providing ample power to operate the servo motor and other components at their maximum capacity. Additionally, the USB micro-B connector is vital for charging the battery. Simply connect an external power source like a USB power adapter rated at 5V@1.5A to start charging.

MOSFET Switch

The Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) switch on the *Classroom Exos* PCB, Figure 5.2 is responsible for controlling the power supply to the servo motor. It acts as an electronic switch that can turn the power on or off based on the control signals received from the microcontroller. In the *Classroom Exos* PCB, we use two MOSFETs to handle power efficiently. The circuit can be observed in Fig 2. The p-channel MOSFET (Si3483CDV) acts as a switch between the battery management input (Source) and the servo motor (Drain). The microcontroller controls the gate of this MOSFET through an n-channel MOSFET (BSS123), ensuring precise control of the power source. This setup improves control, as well as safety and optimizes power usage in the exoskeleton system.

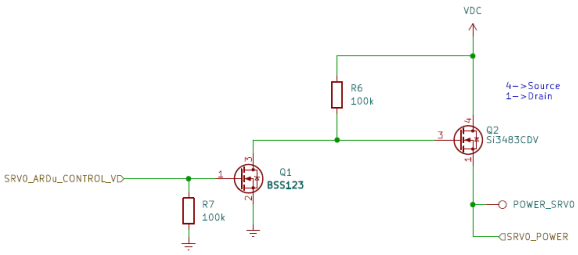


Figure 5.2: *MOSFET Switch (Si3483CDV) in Classroom Exo PCB: Controls power flow from the battery to the servo with microcontroller precision through N-Channel MOSFET (BSS123)*

LED Pushbutton

The Lp11 is a latching non-tactile Red-Green-Blue (RGB) push-button switch integrated to easily switch the power the microcontroller on or off for. With RGB illumination options, you have the flexibility to change button colors and assign specific colors to different functionalities, enhancing user interaction and system feedback.

Test Points

In the *Classroom Exos* PCB, test points are strategically placed locations that allow you to easily measure and monitor signals or voltages at specific points in the circuit. These points serve as convenient checkpoints for testing the functionality of different components and ensuring that the PCB is operating as intended.

Terminal Blocks

The *Classroom Exos* PCB incorporates five terminal blocks, each serving a specific purpose:

- Servo Terminal Block (4 Positions): Connect the servo motor securely using the 4-position terminal block.
- Force Sensor Terminal Block (4 Positions): The 4-position terminal block is dedicated to the force sensor, ensuring a reliable connection.
- sEMG Sensor Terminal Blocks (3 Positions Each): Using two 3-position terminal blocks for the sEMG sensors, providing organized connections.
- Battery Terminal Block (2 Positions): The 2-position terminal block enables a secure connection to the battery.

Terminal blocks play a crucial role in the *Classroom Exo* system, offering a secure and organized method to connect and disconnect external wires. They ensure a reliable interface with the circuit, simplifying the overall wiring process. We used screw-less terminal blocks to make it easy to insert and take out the wires.

5.1.2 Microcontroller

The microcontroller, specifically the Arduino Nano 33 IoT serves as the brain, orchestrating the various components within the exoskeleton. It plays multiple roles by reading signals from sensors, controlling motor movements, monitoring battery percentage and managing serial communication over Bluetooth. The microcontroller is easily programmable and modifiable, making it accessible to users with various levels of programming experience. Furthermore, the microcontroller is equipped with a serial communication interface (as well as a Bluetooth interface), providing communication with other external devices to ensure synchronization and overall smooth operation.

5.1.3 Motor and Angle Sensor

The *Classroom Exo* employs an electric servo motor, specifically the CYS-S0650 55KG model - which is the same servo motor as used in the EduExo Pro. This servo motor includes robust gears and control electronics housed within, providing reliable performance. The servomotor incorporates an integrated potentiometer to precisely measure the rotation angle of the motor shaft. The servo motor simplifies wiring with four color-coded wires:

- Power Supply Inputs: Red and brown wires
- Control Signal Input: Orange wire
- Angle Signal Output: White wire

5.1.4 Force Sensor

The force sensor, which is integrated into the exoskeleton, measures the interaction force between the exoskeleton and the user. It can be used to record the interaction and to control the exoskeleton. When force is applied, the metal beam and attached strain gauges are deformed, changing the electric resistance of the strain gauges. These changes in resistance are measured in a Wheatstone bridge circuit: two of the wires of the sensor are for the supply voltage V_s, and the other two are the output voltage V_{out}, (the sensor signal). The output voltage is then amplified by a factor of 1000 by the INA125 amplifier chip. The chips output connects to the Arduino microcontroller that reads the amplified signal via its internal 10-bit Analog Digital Converter (ADC).

5.1.5 Surface Electromyography (sEMG)

The *Classroom Exo* incorporates two Surface Electromyography (sEMG) sensors to monitor muscle activity in the biceps and triceps. These sensors are seamlessly integrated into the system through dedicated AUX plugs connected to the PCB via terminal blocks. We've chosen the Myoware 1.0 Surface Electromyography (sEMG) sensor for its user-friendly features. The Myoware 1.0 sensors come equipped with built-in circuits that effectively filter, amplify and rectify the muscle signals, ensuring precise readings.

5.2 PCB Assembly

Summary

Through this section, we will explain where and how to get our Printed Circuit Board (PCB) design, how to identify, place and solder the components to the Printed Circuit Board (PCB) at the right place. Finally, we will walk you through manufacturing the upper arm cover that contains the Printed Circuit Board (PCB), servo motor, and the sensor connections. As a bonus we will also show you how to manufacture enclosures for your sEMG sensors.

Our commitment to accessibility led us to choose the open-source application KiCad for designing the PCB of the *Classroom Exo*. This decision ensures that all students and educators can benefit from the improvements made to the exoskeleton.

Accessing the PCB Design

1. Download PCB Design File:
   - Visit our GitHub repository at https://github.com/fabianjust/
   - Locate and download the *Classroom Exo* PCB design file.
2. Printing the PCB:
   - Send the downloaded PCB file to your preferred PCB printing company.
   - Most companies prefer the Gerber file format; you can find it in the "fabrication" folder of our design.

Customization Option
Feel free to tailor the PCB design to your preferences. If you choose to make modifications, create your own Gerber file based on your implemented changes.

Ordering Components
In addition to ordering the PCB, ensure you order the necessary components. Refer to our Excel sheet , providing details on required components, quantities, and links to potential suppliers.

5.2.1 Preparing interactive Bill of Materials (BOM)

For soldering the components to the PCB we will start with creating a Bill of Materials (BOM) from our KiCad file. With the Bill of Materials (BOM) we can then easily identify the PCB components, and check off the list once the components are placed on the board.

Open the *EDUEXO-custom-PCB.kicad_pro* file in KiCad, go to *Plugin and Content Manager* - a new window with the Plugin and Content Manager will open, as you can see in Figure 5.3. There, type *Interactive HTML BOM* in the search bar and install the Plugin.

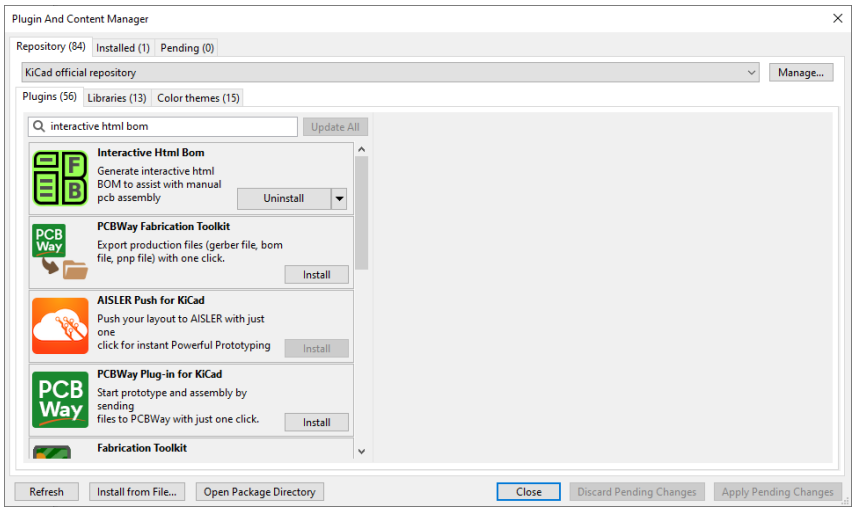
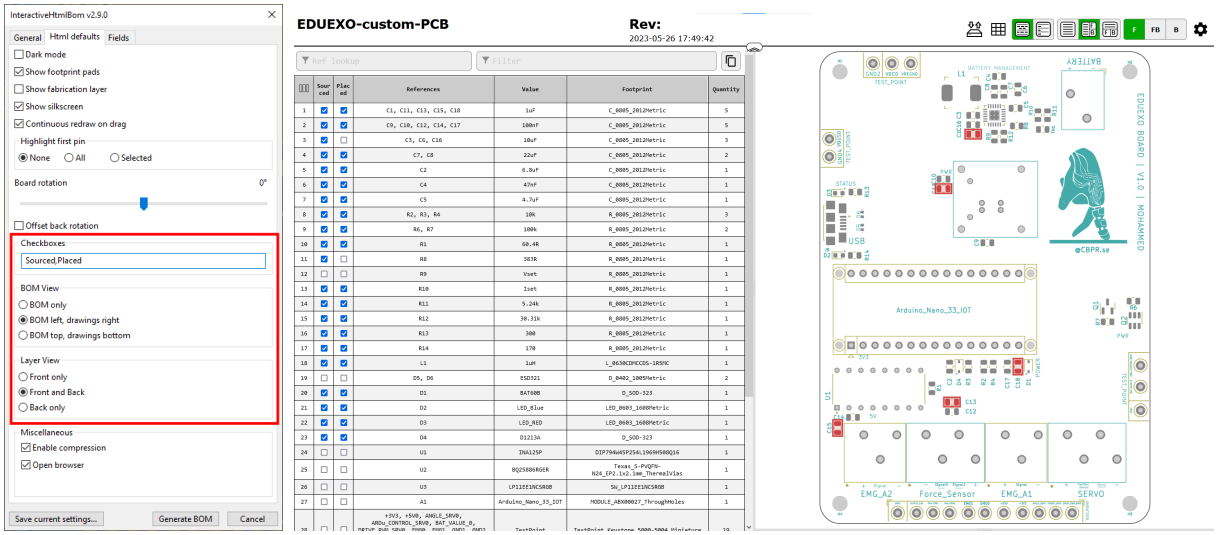


Figure 5.3: *Finding the interactive BOM HTML plugin in the Plugin and Content Manager*

After successfully installing the Plugin, you can open the *EDUEXO-custom-PCB.kicad_pcb*, this shows you the PCB. In the Toolbar click on the light green Icon, this opens a new window for editing the default HTML BOM settings. We only want to change some things in the *HTML defaults* tab, see also Figure 5.4a:

1. *Checkboxes* should list Sourced, Placed
2. *BOM View* should check BOM left, drawings right
3. *Layer View* should check Front only



(a) *Settings for the interactive BOM.*

(b) *Components within the interactive BOM.*

Figure 5.4: *Components within the interactive Bill of Materials (BOM) HTML file*

All the remaining settings can stay the same, but you are free to change more if it suits your needs. Next,

click on *Generate BOM*, this saves the HTML file into the previously set directory - by default the file is saved to the KiCad project directory.

Hint

The HTML file can be opened by any standard browser, though we found Chrome to work best. We don't recommend to use Firefox since there are issues with displaying the file correctly.

The HTML file is interactive: Try hovering over the first component in the list. You will see that the corresponding component is highlighted in the graphic display of the PCB, like in Figure 5.4b. We will use the interactive HTML to keep track of the components we sourced and placed on the PCB, this way we won't mess up the order of components!

5.2.2 Step-by-Step Guide for Soldering Components

Next, we will start soldering our components onto the PCB. Here is al list of all required materials to continue:

- Solder wire, paste and wick
- Flux
- Soldering station
- Tools such as pliers, tape
- Reflow oven or Hot air gun (typically included with the soldering station)
- Multimeter (recommended)
- Isopropyl Alcohol (IPA) for cleaning (optional)
- appropriate Personal Protective Equipment (PPE), such as gloves and protective eye-wear

Stage 1: Applying solder paste
To start off, wipe down your empty Printed Circuit Board (PCB) with some Isopropyl Alcohol (IPA) to remove any dust, dirt and oils. Next, we will apply the solder paste to every surface connection pad on the board. You can either manually push out the solder paste from your syringe, or use a pressurized solder paste dispenser. Manually dispensing the paste is quite tricky, so you will have to figure out the best way to hold the syringe and how much pressure you need.

Hint

We recommend watching some tutorials like **this one** on YouTube before you get started with soldering. You can also expect to need some practice, so don't get discouraged when your first solder joint don't look nice!

In Figure 5.5 you can see the prepared Printed Circuit Board (PCB), where solder paste is applied to all Surface Mount Technology (SMT) connection pads.

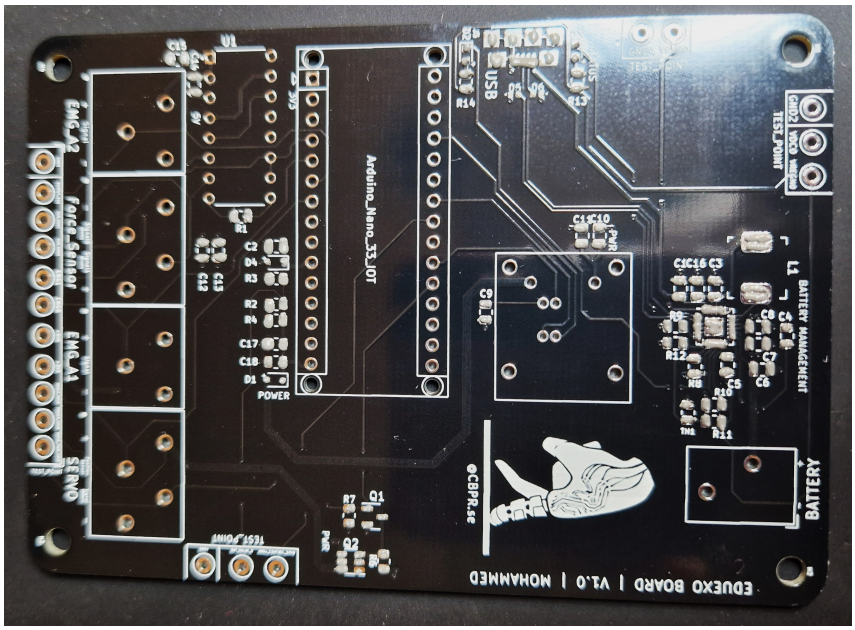
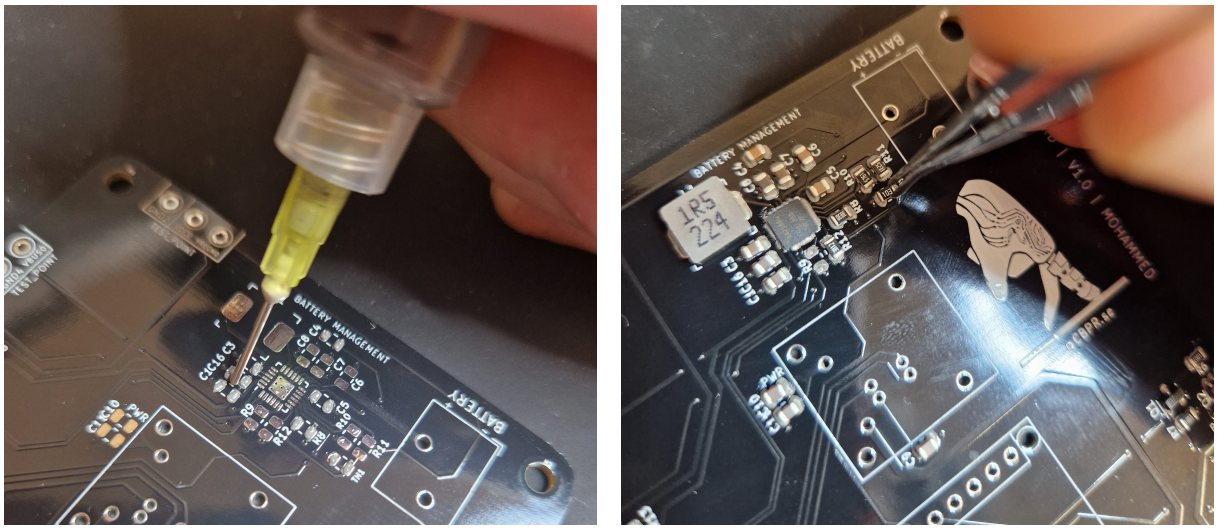


Figure 5.5: *Empty PCB, which was wiped down by IPA and applied solder paste onto the connection pads.*

Stage 2: Placing the Surface Mount Technology (SMT) components

Using the previously created HTML Bill of Materials (BOM), you can now start placing all the SMD components on your board. First, start off with capacitors, resistors and inductors. Next, place the Light Emitting Diodes (LEDs) and other Diodes, but make sure you place them in the correct orientation! Continue with placing the Battery Management Integrated Circuit (IC), and USB port.



(a) *Placing the Solder Paste on the PCB*

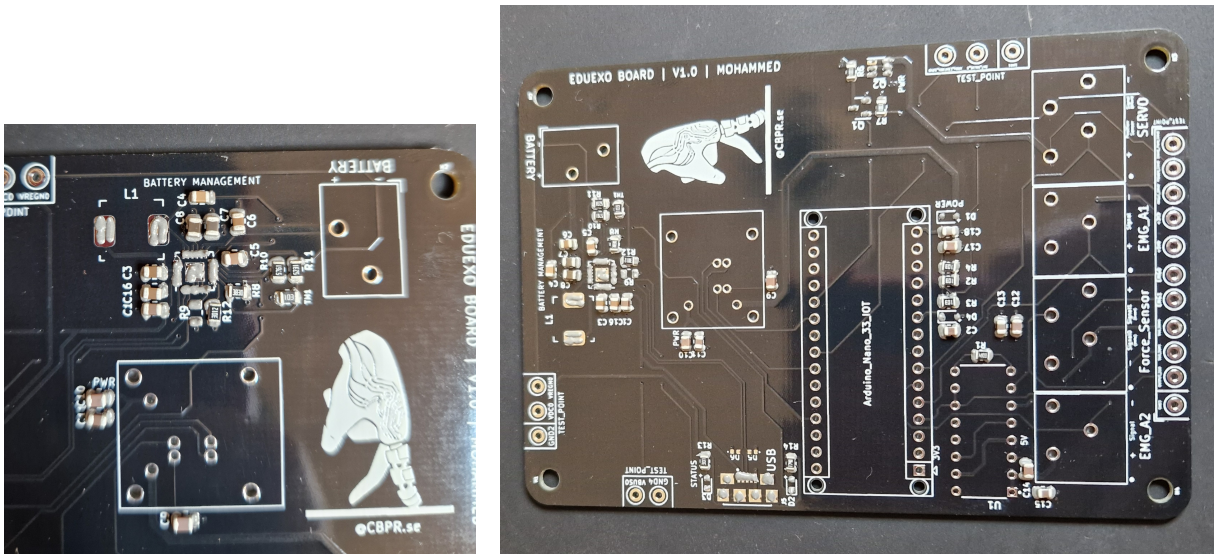
(b) *Placing the SMD components on the PCB.*

Figure 5.6: *Application of Solder Paste and SMD components on the PCB.*

Hint

Keep your hands steady, take breaks to avoid fatigue and use appropriate Personal Protective Equipment (PPE), such as eye protection and proper ventilation.

Now that you have placed all your SMD components like in Figure 5.7b, you can finally solder them to the board. Depending on your equipment, follow either the directions using the reflow oven, or hot air gun.



(a) *Closeup of placed components on PCB* **(b)** *Finished placing the SMD components on the PCB.*

Figure 5.7: *After application of Solder Paste and SMD components on the PCB.*

Stage 3a: Soldering using a Reflow Oven

Using a reflow oven for soldering the placed SMD components is pretty straight forward. You just have to follow the instructions on your oven, place the prepared PCB inside and let the process run.

We used the *Vapour Phase Reflow Oven RF275*, the soldering cycle for that takes about 10 to 15 minutes. Depending on your oven, make sure to only use it in a well ventilated area and use gloves when handling potentially hazardous liquids.

Stage 3b: Soldering using a Hot Air Gun

Soldering the components using a hot air gun, that often comes with a a commercially available solder station, is harder and requires a bit of practice. Set the hot air gun to xx degrees and let it heat up. Slowly in a circular motion move the nozzle of the hot air gun around the area on your PCB where you want to solder the components. We recommend starting with a simpler section, with less components, such as the area around the MOSFET switch. You can help to keep the components steady by using the tip of the pliers you used to place the components beforehand.

Hint

Practice soldering on scrap PCBs before working on your project. Make sure to not touch the surface of the PCB with the nozzle of your hot air gun and adjust the soldering temperature based on the component and PCB requirements.

Stage 4: Soldering Through Hole Components

Now we can solder the first through hole components! Start off with soldering the battery terminal block using a standard soldering iron and solder wire: Place the 2-position terminal block through the holes on the board and secure it's position with some tape. Now turn the PCB around and heat up the connection points for 2 seconds, apply solder and continue heating for 1-2 seconds. Remove the solder iron tip and let your connection cool off.

Hint

Make sure to use enough solder and heat up the connection points well, to avoid so called "cold" solder joints. Cold solder joints can break easily and don't provide a good connection. Your solder joints should be smooth, shiny and a little bit cone-shaped.

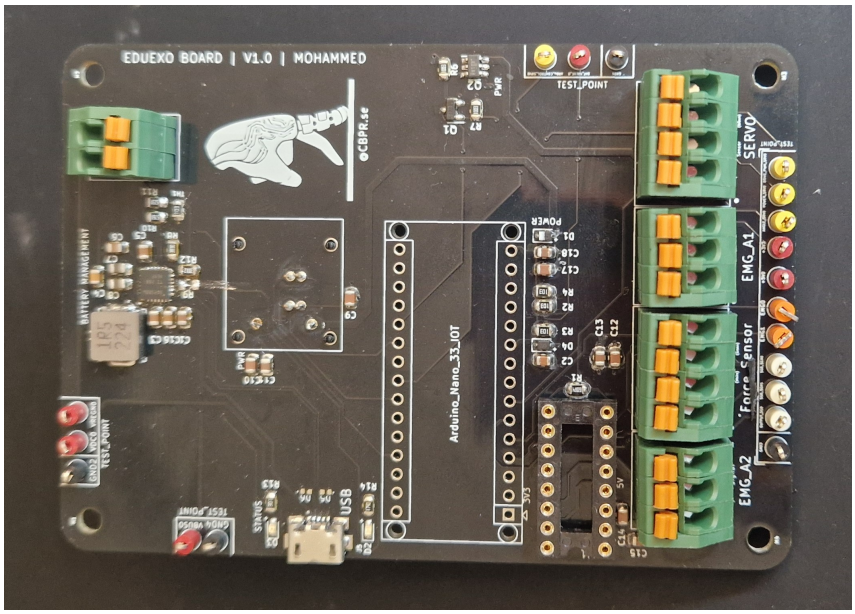


Figure 5.8: *PCBafter applying all SMT components and most through hole components.*

Continue with soldering the test points for voltage labeled VDCO VREGN0, VBUS0 and BAT_VALUE_0 using the red pins. As well as the test points for ground labeled GND1, GND2 and GND4 using the black pins in the same fashion as the battery connector.

Stage 5: Testing component functionality

First, we have to verify that the battery management IC works as intended. For that the safest way is to use a benchtop DC power supply, some connector cables, a multimeter and the battery power connectors on your PCB. Set the current limit on your power supply to 100mA (0.1 A) and set the output voltage to 7.4 V. The low current limit ensures that we don't damage the battery management IC during measurements. Make sure the output from the power supply is turned **off**! Connect the black wire to the negative pole (-) of the battery connector, and connect the red wire to the positive pole of the connector (+). Now, you can turn the output of you power supply on. Next use your multimeter and verify the voltage at the test points in the upper left corner. The measured voltage between VDC0 and GND4 should be at least 6.2 V and between VREGN0 and GND4 xxxV.

Next, we have to check the functionality of the USB port. Turn off and disconnect your benchtop power supply from your PCB. Connect your charging USB cable to the port on the PCB. Make sure both the red and blue LEDs light up, see Figure 5.9. After a short amount of time, you should start to hear a beeping noise and the blue LED will blink. This means the IC recognizes that no battery is connected.

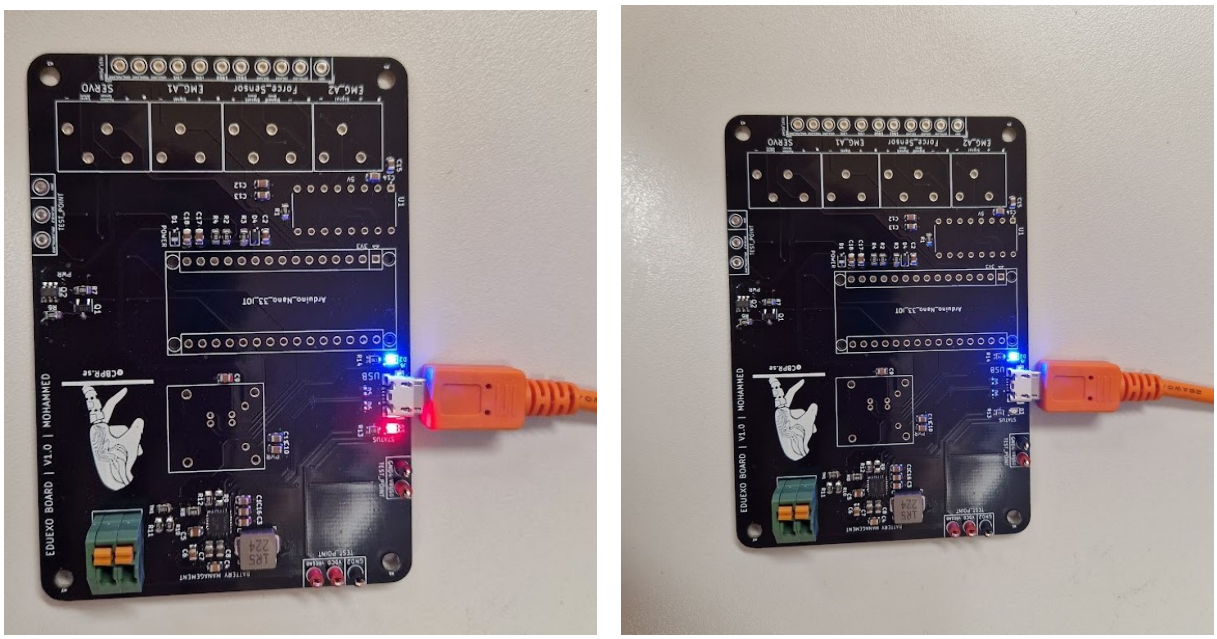


Figure 5.9: *Testing of the USB port on the PCB.*

Finally, disconnect your charging cable and connect your battery to the battery connectors. Make sure to check that you use the right polarity! Now, re-connect the charging cable and you should be able to observe both lights turning on, like in Figure 5.10.

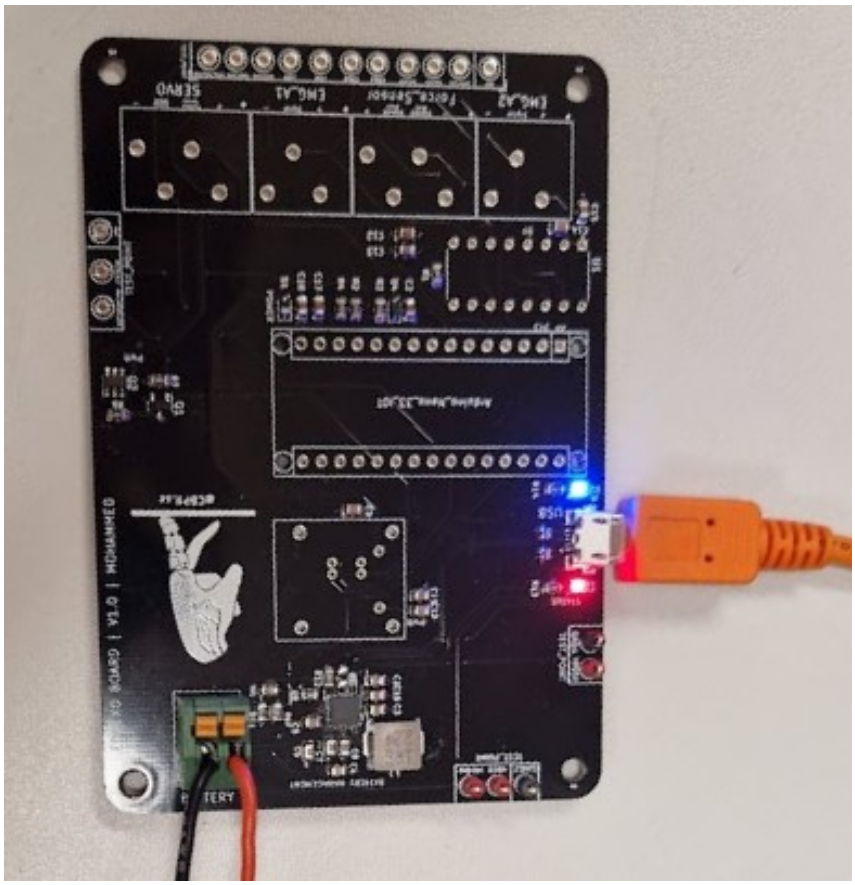


Figure 5.10: *Checking the USB charging port functionality on the PCB.*

Hint

For any issues with the multimeter readings, particularly around the battery management IC, recheck soldering.
Verify that the blue LED doesn't blink during USB charging; if it does, inspect the battery or charging adapter.
Always disconnect the battery after testing to prevent potential damage to the board and components.

Stage 6: Soldering remaining Through Hole Components

After we have verified the functionality of the PCB and battery management IC, we can solder the remaining through hole components.

Using the same methodology as described in Stage 4 (see section 5.2.2) solder the socket for the INA 125 amplifier. Next, solder the strip connectors for the Arduino Nano 33 IoT board as well as the lower terminal blocks.

Now, you can solder the lower test points onto the board. Use white for sEMG sensors, orange for the force sensor, yellow for the servo motor and again red for VDD and black for GND.

Finally, you can complete your board with soldering on the RGB LED Pushbutton on the backside of the PCB. By completing these steps, all components on the bB should be soldered!

Stage 7: Preparing Arduino Nano 33 IoT board

Before you can connect your Arduino board to the PCB, you need to solder the pins that come with the board. Again, use the same methodology as described in Stage 4 to solder the pins onto your board.

Hint

Make sure to only touch the board on it's edges. Do not touch the sensitive SMT components on the board, this can damage the components and destroy your microcontroller!

Stage 8: Placing Arduino Nano 33 IoT board and amplifier

Finally, you can place the Arduino and the INA125 amplifier into their sockets on the Printed Circuit Board (PCB), see Figure 5.11.

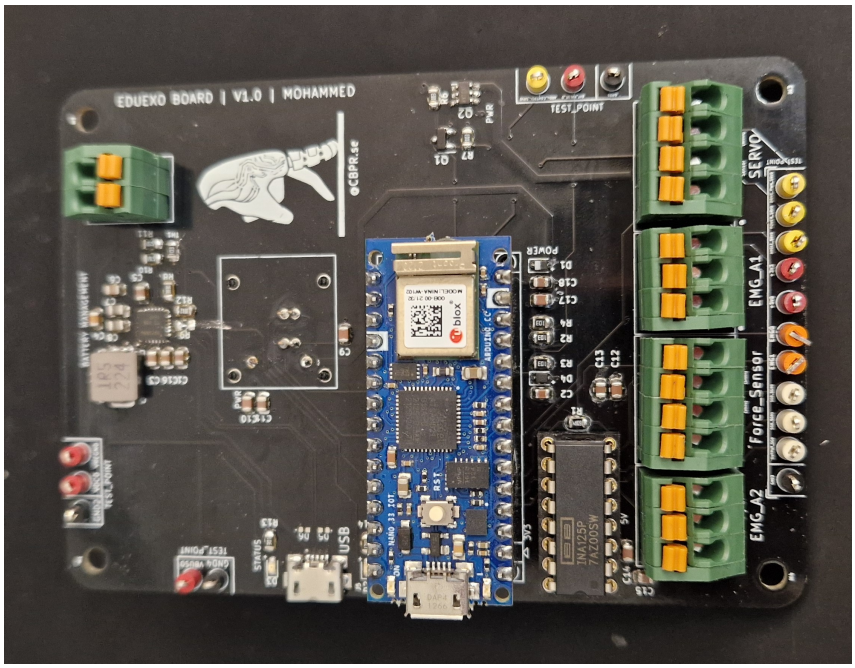


Figure 5.11: *Completed PCB with all its SMT and through hole components mounted.*

5.3 Manufacturing the Enclosure Components

Summary

In this section you will find step-by-step instructions how prepare and manufacture the components for the *Classroom Exo* enclosure. The manufacturing process consists of three main steps. First, we will 3D-print the enclosure components. Next, we will laser cut the front side of the device. Finally, we will print the sEMG enclosure to complete the assembly.

5.3.1 3D Printing the PCB Enclosure

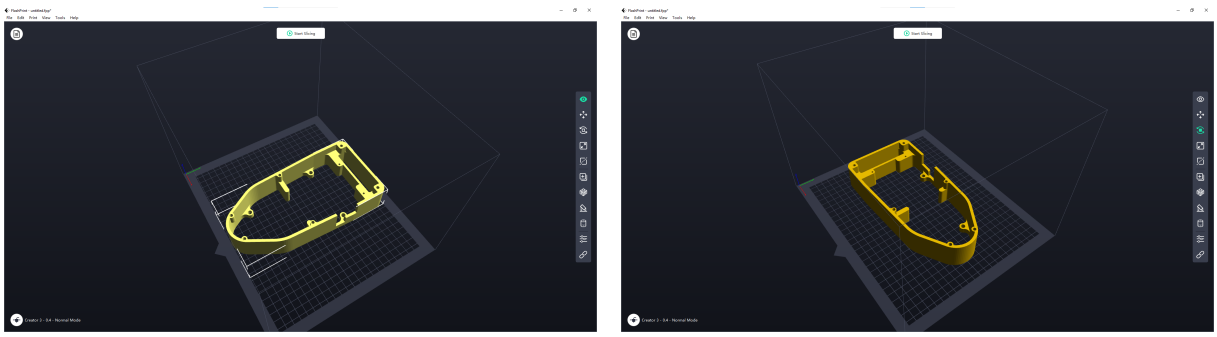
For our custom PCB we have constructed a 3D-printable part, that you need to print yourself! Here we will walk you through that process. First, you will need to download our STL-file *EduExo_Sidewall_v5.STL* from our Github (https://github.com/fabianjust/). This is like a 3D blueprint of what you want to print. It's a computer file that describes the shape of your object using tiny triangles that fit together to form the surface - kind of like wrapping a present with small pieces of paper.

Next you will need to import that file into a slicer software that is compatible with your 3D printer. This is the program that converts your STL file (3D model) into G-code (printer instructions). It literally *slices* your 3D model into many thin layers and figures out how to fill the inside of the model, where to put support structures for overhanging parts, how thick to make the walls and how fast the printer should move. There are many more settings that affect print quality, but we can't talk about all of them here! For our parts we used the *Flashforge Creator 3* printer and it's custom software *FlashPrint 5*.

Attention

If you have the same printer, you can use the *EduExo_Sidewall_v6.gx* instead. This is already the G-code file, which can be uploaded to your printer directly. Only do this if you are confident that you have the **exact** same printer (*Flashforge Creator 3*), otherwise you could damage your printer!

The Side Wall Segment is quite large, with dimensions of 220 mm x 110 mm x 37.5 mm. Make sure to orient the model on your platform, so it fits comfortably. Generally you want to avoid coming within 1 or 2 cm of the edge of the printers build plate. You can see how we rotated the model around it's z-axis in Figure 5.12a and Figure 5.12b, to be further away from the build plate edges!



(a) *Incorrect placement of the model.* **(b)** *Model rotated 90˚ around the z-axis.*

Figure 5.12: *Examples of object placement on build plate within the slicer software.*

In Figure 5.13 you can see the settings we used for printing with Polylactic Acid (PLA). Depending on your printer, and the material you are using these may differ. For our printer the model took 8.5 hours, and used ca. 100g of PLA.

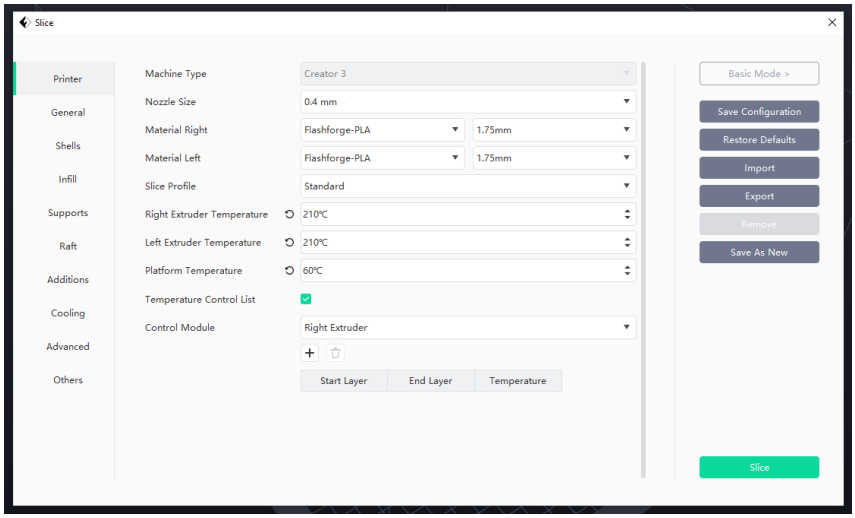


Figure 5.13: *Settings for printing with PLA on the Flashforge Creator 3.*

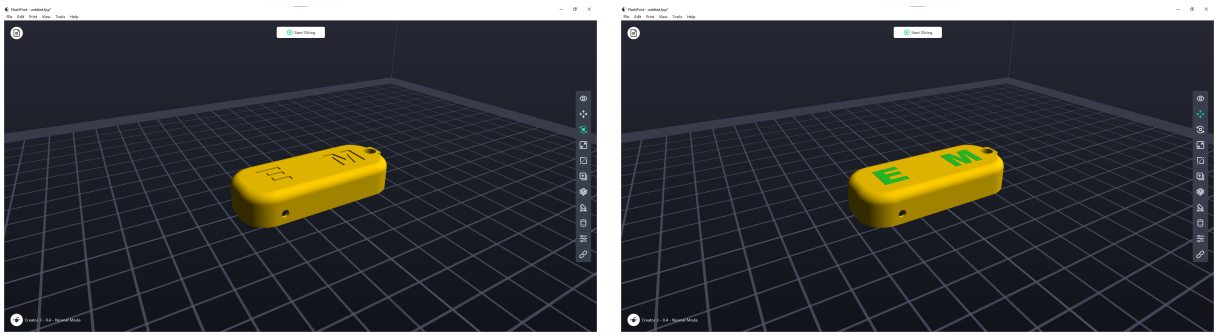
Hint

If you are not sure which settings you should use with your printer, we recommend trying printing some benchys. Play around with the temperatures, layer heights and print speeds to find what works for you. There a lots of tutorials online, to help you with that!

5.3.2 3D-prinitng the sEMG-sensor enclosure

If you want to use the Myoware 1.0 sEMG sensors, we have created a simple sensor enclosure, to protect it from damage. Again, the stl.files can be found on our GitHub: *Myoware_1.0_Case_V1.stl* and *Myoware_1.0_Case_V1-letters.stl*

You will see that there are two separate models. Since our printer is able to print with two materials, we took the liberty to create a colorful enclosure. Import the main model into your slicer software, place it in the center like in Figure 5.14a . Then, import the second model and make sure to place the letters into the free space in the main model, see Figure 5.14b. Since the letters have a thickness of 5 mm, you can subtract that from the height of the main model. This means you need to place the letters at a height of 10.2 mm!



(a) *Centered placement of the main model.*

(b) *Centered placement of the letters, at a z-offset of 10.2 mm.*

Figure 5.14: *Examples of object placement on build plate within the slicer software.*

Hint

Not all 3D printers are capable of printing with multiple materials or colors. If yours is not able to do that, don't worry! Just print the main file without the additional letters!

In your print settings you will have to select the main model for the right extruder and the second model for the left extruder. To do that, select the model and klick right - in the settings you will select the option *Choose Right Extruder To Print* and *Choose Left Extruder To Print* respectively. For our enclosures, we used black PLA for the main body, and red PLA for the letters.

Attention

If you have the same printer, you can use the *Myoware_1.0_Case_V1.gx* instead. This is already the G-code file for 4 models, which can be uploaded to your printer directly. Only do this if you are confident that you have the **exact** same printer (*Flashforge Creator 3*), otherwise you could damage your printer!

Before slicing the object, we suggest adding a wiping tower. This ensures that the color change won't be messy! You will also need to add supports for the main body, since the roof is overhanging more than

40°! With the same settings as in Figure 5.13, our model took around 1 hour to print, with around 7.7g of total material used.

5.3.3 Laser cutting and engraving the cover

We are so proud of our custom PCB, we wanted to make it the heart of the *Classroom Exo*. To do this we decided to go for a naked design, which showcases the internal parts of the *Classroom Exo*.

For this we used 3mm Acrylic plates, and cut out the front side part with a laser cutter. We also added some labels and icons, where we used the engraving option of the laser cutter. You can use the *eduexo_frontplate.dxf* to set up your laser cutter. We won't go into detail how to create the front-side cover in a laser cutter, make sure to use the correct software and recommendations for your specific device!

Hint

If you don't have access to a laser cutter you can also just print the front plate with your 3D printer. Just keep in mind that the internal parts won't be visible that way.

5.3.4 Soldering the sEMG sensors

After we have successfully printed the covers for the sEMG sensors, we need to solder a suitable cable to the sensor board.

Since you already assembled the Printed Circuit Board (PCB) on your own, you should have all your materials for soldering the Surface Electromyography (sEMG) sensors!

You will need

- Myoware 1.0 sEMG sensor
- 3D printed enclosure
- Audio cable
- Solder wire, and wick
- Flux
- Soldering station
- Tools such as pliers, tape
- Multimeter (recommended)
- Isopropyl Alcohol (IPA) for cleaning (optional)
- appropriate Personal Protective Equipment (PPE), such as gloves and protective eye-wear

Hint

Make sure to gather all the needed materials before you start soldering your sensor connections!

Verifying cable polarity

As a first step, we recommend to plug the sensor cable into one of your jack port . Using a Multimeter you should now verify which cable is connected to the red, white and black cable on your jack plug. In our case the sensor cable hat a brown and white cable inside, which were connected to the red and white cable on the jack port, with the ground consisting of the loose copper threads.

Pre-Assembling the sEMG

Before soldering the connections make sure to insert the sensor cable through the hole on the top of the printed enclosure.

Soldering the connections

Start by cutting the entire sensor cable to your desired length, we decided to cut the cable to 20cm.

Hint

You don't have to shorten the sensor cables, but we found that a shorter cable helps with signal quality and ease of use!

Now, strip the insulation off of the outer most layer of your cable for a length of 10 mm. Next, strip the insulation of the two inner cables - the brown and white for a length of 5 mm. With the ground wires, create a small bunch with an approximate diameter similar tor the other two cables. Cut off the remaining ground wires.

After stripping the insulation, twist the copper threads of each wire together and coat the tip with some solder. Now referencing the cable polarities from the previous section, insert the prepared wire tips into the through hole solder points on the sensor board.

Making sure to heat up the solder joints, and using the appropriate amount of solder, connect each cable to the sensor board. Using a multimeter verify the connections.

Preparing the reference cable

When using the Myoware 1.0 sEMG sensor you will need a reference cable. For each sensor there is one reference cable provided, but the can be quite short (5 cm).

For this reason we re-connected the end of the reference cable, which gets plugged into the board, to a longer cable with a snap-fit connector at the end. That way we are free to connect the cable to a bony reference point, further away from the muscle bulge.

Just, cut off the distal end of the reference cable with about 2 cm length. Strip the wires, and prepare them by twisting and applying solder. Do the same thing for the longer cable. Connect the two cables together, by twisting them around each other and soldering the wire tips. Don't forget to use a shrinking tube for isolation!

Finishing the sensor board

Now, you can insert the reference cable through the circular hole on the enclosure side below the "E". Plug in the connector, to it's designated place on the board.

> **Hint**
>
> For a secure connection we fixed the soldered joints using a hot glue gun. But we recommend only doing this after you have verified that the sensors are working as they should.

Thread the sensor cable through the hole, so that the sensor board can be inserted into the enclosure. We created a snap-fit enclosure, so you can just clip the sensor board in. If you find it hard to to by hand, you can use a small screw driver to stretch the side where te on/off switch is located.

5.3.5 Spare Parts

In case the parts, where the *Classroom Exo* is attached to your body break or become loose, we have created 3D models for you to print at home!

Just use the stl-files *Cuff_v2.stl* and *Wrist_Cuff_v2.stl* from our GitHub and prepare them for printing. Since those cuffs are the parts where your lower and upper arm are attached to the *Classroom Exo*, we recommend to use a higher infill setting (i.e. 30 to 40%). This means that the print will take a lot longer, but the finished part will also be more sturdy. As for materials we used PLA, but more sturdy materials like Acrylonitrile Butadiene Styrene (ABS) or Polyethylene Terephthalate Glycol (PETG) would likely work even better.

5.4 Assembling and Connecting

Summary

In this section you will find step-by-step instructions how to assemble the upper arm cover, and connect the external components like the sensors and servo motor to the PCB. Follow along to continue our *Classroom Exo* assembly!

After successfully completing the PCB you will move forward with assembling the upper arm cover and connecting the external components, such as the sensors and servo motor, to your PCB.

5.4.1 Pre-Assembling the Upper-arm Cover

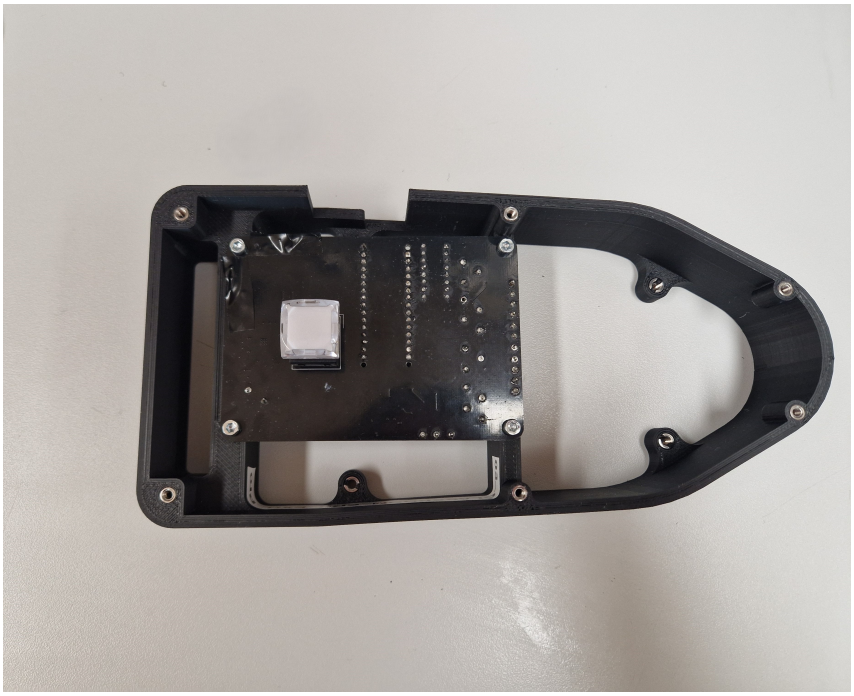


Figure 5.15: *The Sidewall segment has 10 "M4" holes and 4 "M5" holes.*

Assembling the Sidewall

With all the parts ready, you can now assemble the upper arm cover of the *Classroom Exo* by inserting the screw thread inserts into the holes of the sidewall segment. You'll need 10 M4 screw thread inserts and 4 M5 screw thread inserts for this step. Insert a screw into the thread insert, then screw the thread insert into the hole using the tip of the inserted screw. Repeat this process for all the required inserts.

Next, flip the PCB upside down, so that the pushbutton faces upwards. Use the M3 screws on the 4 corners to attach the PCB to the Sidewall segment.

Assembling the Frontside

Once the sidewall part is assembled, insert the sEMG sensor jack sockets into the frontside segment. Connect the frontside segment to the sidewall segment using the M4 screws.

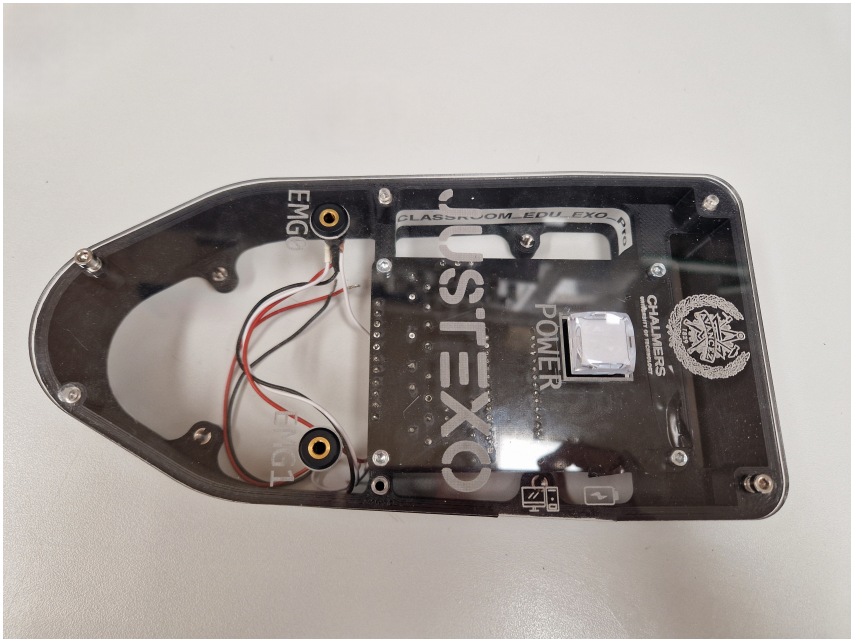


Figure 5.16: *Placing the EMG Plug socket on the Frontside segment of the Upper-arm Cover*

5.4.2 Connecting Motor and Sensors

In this section we will describe how to connect the external components such as the servo motor and all the sensors to your PCB.

Connecting Servo Motor

First we start with connecting the Servo motor to the terminal blocks at the lower end of the PCB. For a visual explanation you can refer to Figure 5.17.

- Power Supply Connection
  - Connect the motors red wire to the terminal block's pole labeled (+)
  - Connect the motors brown wire to the terminal block's ground pole (-)
- Angle Sensor Connection
  - Connect the white wire of the potentiometer angle sensor to the pole labeled *Position Sensor*
- Position Control Connection:
  - Connect the control signal wire (orange) to the pole labeled *Signal*

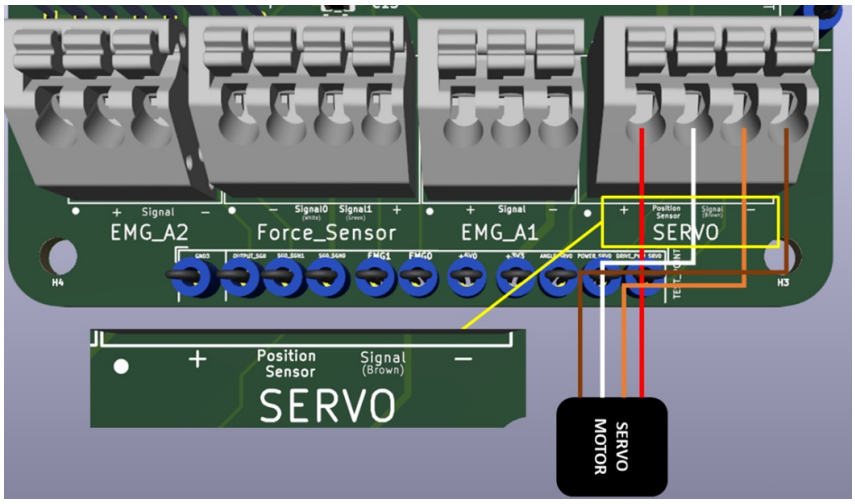


Figure 5.17: *Servo Motor Connection: Visual guide on connecting the servo motor to the terminal block*

Connecting Force Sensor

After connecting the servo motor, proceed to install the force sensor to its designated terminal block on the PCB, according to Figure 5.18. The wire of the force sensor is delicate, so it's advisable to connect it to the 4-wire cable first. For detailed instructions, you can refer to Chapter 5 of the EduExo Pro manual, which provides a comprehensive guide on how to handle this process.

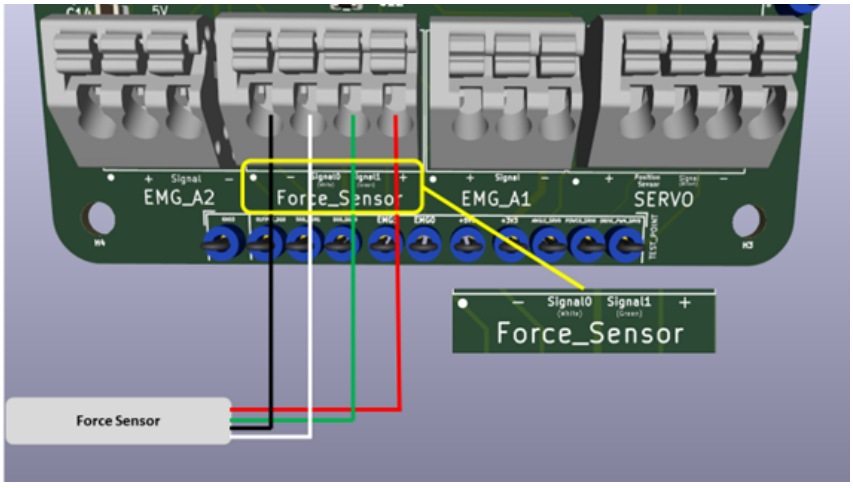


Figure 5.18: *Force Sensor Connection: Follow the color-coded traces to match force sensor wires from the 4-wire cable to their designated terminals for a secure and accurate connection.*

The depicted colors in Figure 5.18, correspond to the force sensor wire colors. However, when connecting the force sensor to a 4-wire cable initially, the colors may differ. To ensure accuracy, trace each force sensor wire from the cable to its corresponding terminal based on the color code presented in the figure. This step ensures a proper match and secure connection. Now, we will explain how to connect the force sensor to the terminal using the force sensor wire colors.

- Power Supply Connection
  - Connect the red wire to the pole labeled (+)
  - Connect the black wire to the (-) pin
- Sensor Cable Connection
  - Connect the white wire to the pole labeled *Singnal0*
  - Connect the green wire to the pole labeled *Singnal1*

Connecting sEMG Sensor

As mentioned before, the exoskeleton has two ports for two external sEMG sensors which are connected to the PCBs terminal blocks.

You should already have inserted the jack ports into the front side of your enclosure. Next make sure to connect the read cable to the (+) icon, the white cable to the *Signal* and the black cable to the (-) icon. You can reference the connections in Figure 5.19 for a visual representation.

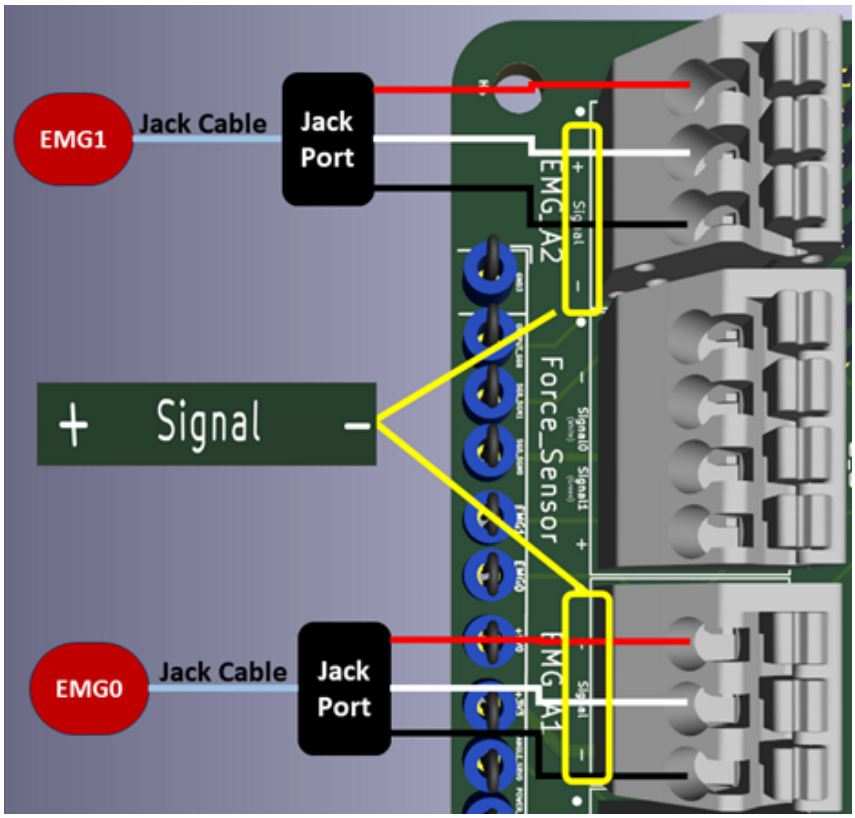


Figure 5.19: *sEMG Sensor Connection: EMG Sensor Wiring: Schematic illustrating the connection of sEMG sensors to a jack socket, with clear guidance on linking the jack socket wires to the corresponding terminals on the PCB*

Now your connected Printed Circuit Board (PCB) should look like the one in Figure 5.20.

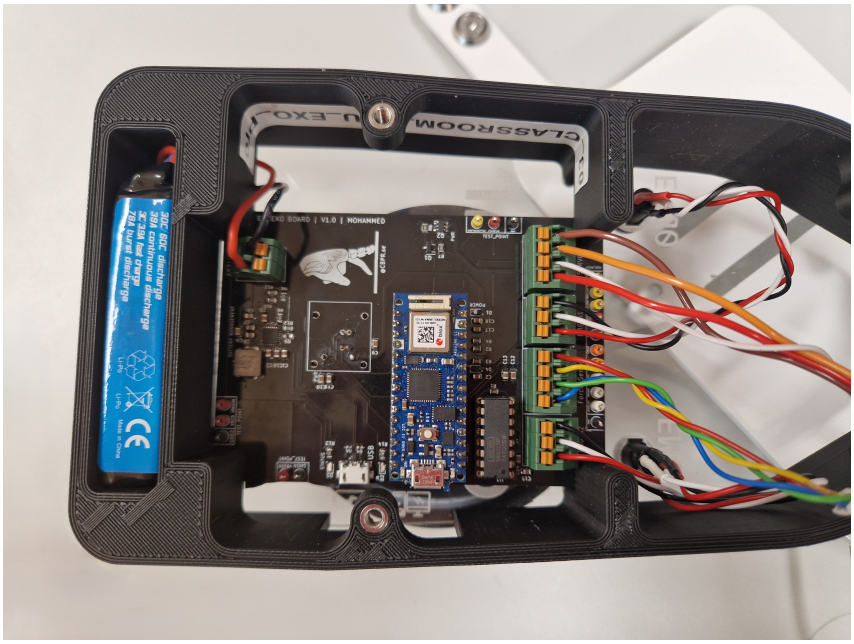


Figure 5.20: *Fully connected PCB which was placed inside the Upper Arm enclosure.*

5.4.3 Finishing the Upper Arm Assembly

After connecting all the wires to the PCB, you will need to finish the assembly of the upper arm cover.

Assembling the Sidewall Segment to the Metal Backside of the Upper-arm Cover
At this point, you should have assembled most of the mechanical parts, including the metal segment with the attached servo, serving as the back side of the upper- arm cover, as depicted in Figure 5.21.

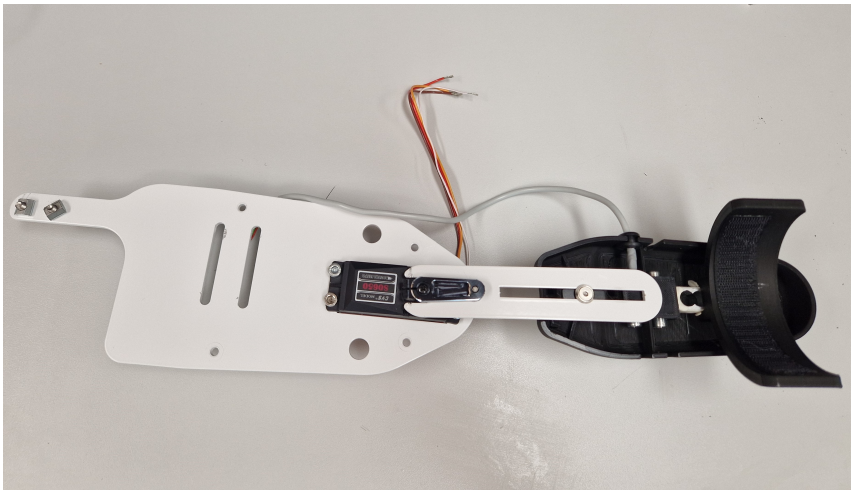


Figure 5.21: *Mostly assembled Upper Arm cover from the backside.*

Screw the backside segment to the sidewall segment using the M5 screws. Now your Upper Arm Cover should look like ours in Figure 5.22.

Figure 5.22: *Fully assembled Upper Arm cover from the backside.*

Congratulations, you have now completely assembled the *Classroom Exo* and are ready to go! Follow the next steps in our tutorial, to test the components.

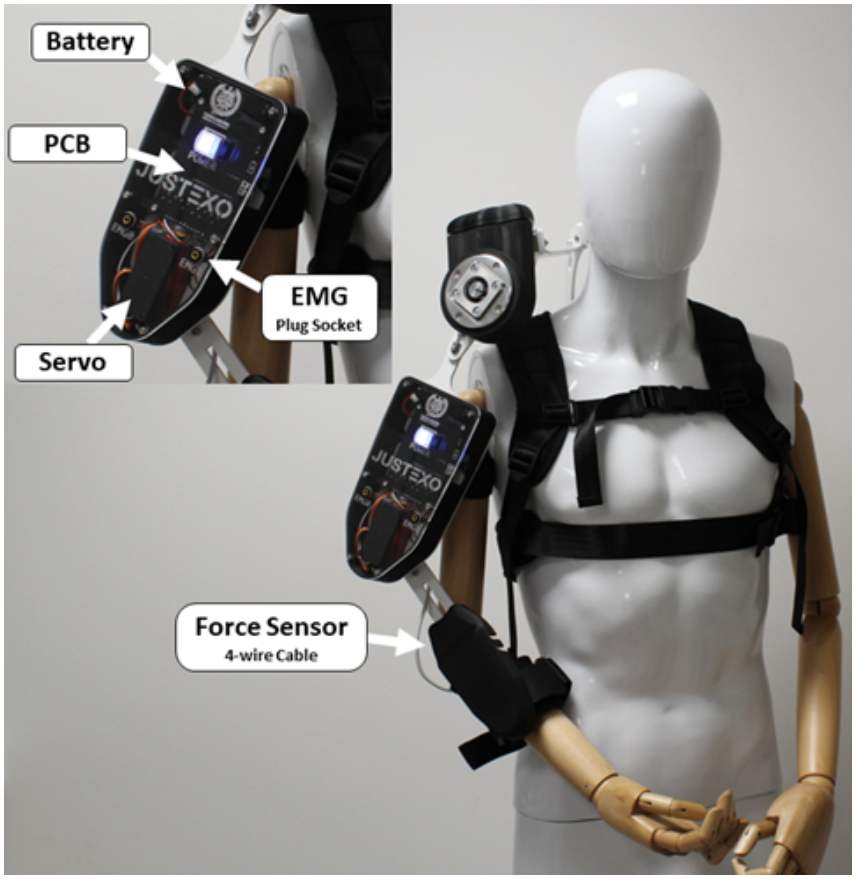


Figure 5.23: *Classroom Exo: A glimpse of the fully assembled Classroom Exo, embodying improved electronics and enhanced functionalities.*

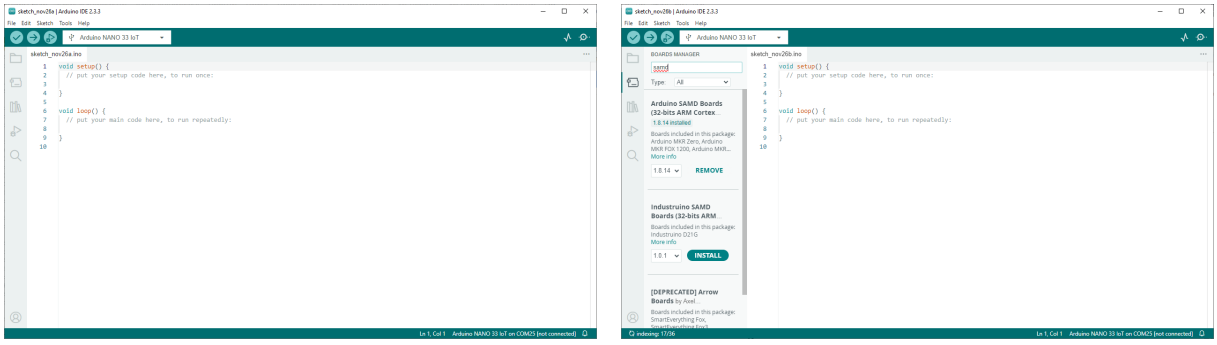
5.5 Component Tests

Summary

In this section we will use the functionality of the PCB we will and focus on the remaining hardware components, such as the servo motor, force and sEMG sensors. The components are tested with short Arduino Code snippets, that are available online: https://github.com/fabianjust/. We will make sure to explain what the code is doing, and which behavior you should expect.

5.5.1 Installing the Arduino IDE

Before you can start programming the micro controller, you have to install the Arduino Integrated Development Environment (IDE). You can download it for several operating systems on the Arduino website: https://www.arduino.cc/. When you install and open the IDE in Figure 5.24a, you will find everything you need to write code and upload it to the Arduino board. To program the Arduino microcontroller, you have to connect it to your computer with a USB A-B cable. You can refer to this link here to get the files to check the servo and sensor using Arduino.



(a) *We use the Arduino IDE for programming all the Components Tests.*

(b) *Make sure to download the correct package for your board.*

Figure 5.24: *Glimpse of the Arduino IDE and the Board manager*

Hint

For more advanced users we recommend using Visual Studio Code with the PlatformIO IDE extension, which offers more advanced debugging options for more complex programs. Please refer to the PlatformIO setup guide here: https://platformio.org/install/ide?install=vscode

Before being able to upload to component tests onto your board, you will have to download the correct board package. For the Arduino Nano 33 IoT you will need to install the *Arduino SAMD Boards* package in the Boards manager.

5.5.2 Testing the Servo Motor

Beginning with the servo motor we want to make sure that we can move it in the defined Range Of Motion (ROM) of 0° to 110°. We will never exceed the 110°, due to hardware limitations: when flexing the elbow joint, you can see that it is possible to hit the 3D printed enclosure with the wrist cuff. This can lead to destroying either part of the enclosures mechanically. Another risk is stalling of the servo motor, which will damage the gears and controller IC of the motor.

The Arduino Code in Listing 5.1 is a short code to move the servo motor from 0° to 110°, and back to 0° again. Note that in line 12 we move the position to 180° instead of 0°. This is due to the orientation of the servo motor in the exoskeleton.

Listing 5.1: *Testing Servo Motor Operation*

```
1  #include <Servo.h>
2
3  Servo myservo;        // create servo object to control a servo
4  int servo_control_pin = 5;
5  int pos = 0;          // variable to store the servo position
6
7  void setup() {
8    pinMode(2, OUTPUT);
9    digitalWrite(2, HIGH); // LOW means Servo is OFF // HIGH
10   delay(20);
11   myservo.attach(servo_control_pin);  // attach the servo on pin 9
12   myservo.write(180);
13
14   for (pos = 0; pos <= 70; pos += 1) { // goes from 0 to 110 degrees
15     myservo.write(180 - pos);          // tell servo to go to 'pos'
16     delay(20);                         // waits 15ms for the servo
17   }
18   for (pos = 180; pos >= 70; pos -= 1) { // goes from 110 to 0 degrees
19     myservo.write(180-pos);            // tell servo to go to 'pos'
20     delay(15);                         // waits 15ms for the servo
21   }
22   myservo.write(180);
23 }
```

We use *myservo.attach()* in line 11 to attach the servo motor on the digital PWM pin 9. Controlling the position of the servo motor is handled by the *¡servo.h¿* library, which is already provided within the Arduino environment. There are two commands we could use *servo.write()*, where we use a value in degree steps, or *servo.writeMicroseconds()*, where we can specify the pulse width in µs. Using the latter option is useful when we want the motor to move with more precision.

Hint

There are different kinds of servo motors for hobbyists, like ourselves. Servo motors already come equipped with a PID controller inside, that controls either the position or the speed of the motor. For our project we used the same hardware, that already came with the Auxivo Edu Exo - which is a servo motor with positional control. So the pulse width equals a discrete position, that the motor will move to. Standard servo motors operate on a control signal with a set frequency of 50 Hz.

5.5.3 Testing the Force Sensor

Next, we will test the functionality of the force sensor, which is located in the wrist cuff of the exoskeleton. We can implement the program that will read the force sensor and display the measured value on the screen using the serial monitor, refer to the code in Listing 5.2. The amplified signal is connected to the analog input pin A0, which can be read using the *analogRead()* command in line 9. Most Arduino boards are equipped with a 10-bit ADC on their analog input pins, we are lucky though, that the Arduino Nano 33 IoT has a 12-bit ADC. This means, we have a higher resolution of our sensor values - with *analogReadResolution(12)* in line 6 we can tell the microcontroller to use the higher resolution on the *analogRead()* commands. The *Serial.print()* command in lines 11 and 12 display the measured values to the Serial Monitor. With the *delay()* in line 13, we pause the code 100 ms until we start in the *loop()* again.

Listing 5.2: *Testing Force Sensor Operation*

```
1  int forceAnalogPin = A0;
2  float forceIs = 0;
3
4  void setup() {
5    Serial.begin(9600);
6    analogReadResolution(12);
7  }
8
9  void loop() {
10   forceIs = analogRead(forceAnalogPin);
11   Serial.print("Force:");
12   Serial.println(forceIs);
13   delay(100);
14 }
```

Hint

Delays are not a good way to program on embedded systems, since they completely block the code execution for the amount of time that is specified. In some Arduino examples you will see delays of 20 up to 2000 or even 3000ms. We will explain how we can write non-blocking code and avoid delays completely in subsection 5.7.4.

5.5.4 Testing the sEMG Sensor

With the sEMG jack sockets connected to the Myoware 1.0 sensor boards, we can now proceed to implement the program that reads the sEMG sensor and displays the measured value on the screen using the serial monitor. Reading the sEMG data is akin to reading the force sensor. The amplified signal is connected to an analog input pin A1 or A2, and the corresponding code for this operation is depicted in Listing 5.3.

Listing 5.3: *Testing sEMG Sensor Operation*

```
1  int emgAnalogInPin2 = A2;
2  int emgSignal2 = 0;
3  int emgAnalogInPin1= A1;
4  int emgSignal1 = 0;
5
6  void setup() {
7    Serial.begin(9600);
8    analogReadResolution(12);
9  }
10
11 void loop() {
12   emgSignal2 = analogRead(emgAnalogInPin2);
13   Serial.print("emgSignal2:");
14   Serial.println(emgSignal2);
15   delay(20);
16   emgSignal1 = analogRead(emgAnalogInPin1);
17   Serial.print(",");
18   Serial.print("emgSignal1:");
19   Serial.println(emgSignal1);
20   delay(100);
21 }
```

In line 12 and 16 we read the amplified sEMG signal from the analog pin using *analogRead()*. Displaying the data in the Serial Monitor happens again in lines 13 to 14 and 17 to 19. Since we will plot two signals simultaneously, instead of watching the Serial Monitor we can also use the Serial Plotter. This displays our sEMG signals in a graph. Now if you flex your muscles, you should be able to see an increase in the analog readings. If you relax your muscles, you should see the signal diverge towards zero.

5.5.5 Testing the LED Button

Finally, we will make sure our LED pushbutton works as intended. Since the button as an RGB LED inside, there are three pins we have to talk to. The intensity of each LED can be set by a PWM signal, this means we will adjust the duty cycle of our signal to get the right color.

Let's see how this works in the example to the red LED: we will use the *analogWrite()* command to set the brightness of the LED. analogWrite() works in the range from 0 to 255, with 0 setting the PWM to 0% duty cycle - meaning the output is **LOW**. The value 255 sets the duty cycle to 100%, turning the output to **HIGH**.

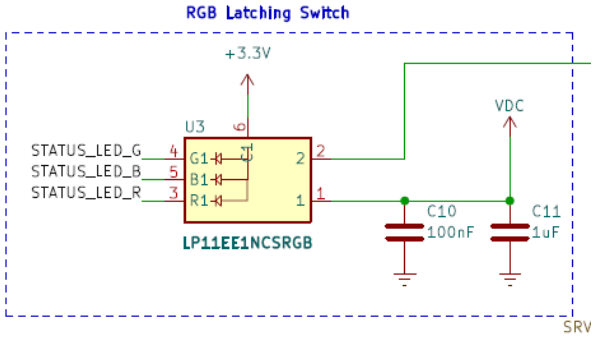


Figure 5.25: *LED Switch in Classroom Exo PCB: Controls power flow from the battery to the microcontroller.*

Now, it may be a bit confusing, but in our configuration analogWrite(PIN_RED, 0) turns the red LED on! Why is that? Let's look at the circuit in Figure 5.25: the LEDs are connected to the Arduino Nano 33 IoT pins in *current sink mode*. This means, when the output is set to **LOW**, current is flowing from V3.3 through the LED into our pin, bringing the LED to light. When the output is set to **HIGH**, no current is flowing and thus the LED is not lighting up.

Since our configuration is the opposite of the standard RGB color code, you will have to invert the values to get the desired color. Listing 5.4 shows a script for testing the LED button, with three different colors each showing for three seconds.

Listing 5.4: *Testing RGB LED Button Operation*

```
1  const int redPin   = 9;
2  const int greenPin = 6;
3  const int bluePin  = 3;
4
5  void setup() {
6    pinMode(redPin,   OUTPUT);
7    pinMode(greenPin, OUTPUT);
8    pinMode(bluePin,  OUTPUT);
9  }
10
11 void loop() {
12   // green
13   analogWrite(redPin, 255);
14   analogWrite(greenPin, 0);
15   analogWrite(bluePin, 255);
16   Serial.println("green");
17
18   delay(3000); // keep the color 3 seconds
```

```
19
20      // blue
21      analogWrite(redPin, 255);
22      analogWrite(greenPin, 255);
23      analogWrite(bluePin, 0);
24      Serial.println("blue");
25
26      delay(3000); // keep the color 3 seconds
27
28      // red
29      analogWrite(redPin, 0);
30      analogWrite(greenPin, 255);
31      analogWrite(bluePin, 255);
32      Serial.println("red");
33
34      delay(3000); // keep the color 3 seconds
35
36  }
```

5.6 Calibrating the Classroom Exo

Summary

Before you can use your *Classroom Exo*, you will need to go though some calibration procedures. In the previous section, we made sure the components work. In this section, we need to make sure to calibrate each component, so the control software will work correctly!

5.6.1 Calibrating the Force Sensor Offset

First, we will need to figure out the *default* force sensor value or *zero force offset*. For that you can use the code in Listing 5.5.

Listing 5.5: *Calibration of Zero Force Offset*

```
1  const int forceAnalogPin = A0;
2  int forceIs;
3  const int rep = 100;
4  int counter = 0;
5
6  void setup() {
7    Serial.begin(9600);
8    pinMode(2, OUTPUT);
9    // Set ADC resolution to 12 bit
```

```
10    analogReadResolution(12);
11  }
12
13  void loop() {
14    forceIs = analogRead(forceAnalogPin);
15    Serial.print("Force: ");
16    Serial.print(forceIs);
17    delay(20);
18
19    counter++;
20
21    if(rep == counter)
22      exit(0);
23  }
```

This program, reads our force sensor 100 times, prints each reading and waits 20 ms between readings. After completing 100 readings the program stops.

You can now use the average of the offset readings and write the value down.

5.6.2 Calibrating the Servo Motor Positions

Next, we have to find the potentiometer sensor value to it's corresponding motor position, see the code in Listing 5.6.

Starting at line 17, the program begins its main operation with a loop that runs 5 times. Within each iteration, there's another loop starting at line 18 that controls the servo's movement. This inner loop starts the servo at 180 degrees and moves it down to 70 degrees in steps of 10 degrees.

At each position in line 20, the program moves the servo to the new angle and waits 1 second. Then in lines 23-228, it takes a reading from the analog pin, prints both the reading value and the current angle to the Serial Monitor, and waits another second before moving to the next position.

Listing 5.6: *Calibration of Servo Motor Positions*

```
1   #include <Servo.h>
2
3   Servo myServo;
4   int servo_control_pin = 5;
5   int ServoAnalogInPin = A3;
6
7   void setup() {
8     Serial.begin(9600);
9     delay(1000);
10    analogReadResolution(12);
11
12    pinMode(2, OUTPUT);
```

```
13      digitalWrite(2, HIGH); // turn on Servo Power!!
14      myServo.attach(servo_control_pin);
15      delay(1000);
16
17      for(int i=0; i<5; i++){
18        for(int angle = 180; angle >= 70; angle -= 10) {
19          myServo.write(angle);
20          delay(1000);
21
22          int potValue = analogRead(ServoAnalogInPin);
23          // Serial.print("Pot value: ");
24          Serial.print(potValue);
25          Serial.print("␣");
26          // Serial.print(" for angle: ");
27          Serial.println(angle);
28          delay(1000);
29        }
30      }
31    }
32
33    void loop() {
34    }
```

The empty loop function at the end (lines 34-35) means that after completing all 5 cycles of measurements, the program simply stops and does nothing further. Each complete cycle tests 12 different positions and takes about 24 seconds. This meas the the total runtime takes approximately 2 minutes.

After this you can take the average for each position of the 5 iterations and save them for the next step.

5.6.3 Adding the calibration values to you Software

Now you will need to use the calibration values for your force sensor and servo motor and insert them into the correct place on the embedded code. Go to the file *config.h* and change the settings like in Listing 5.7.

Listing 5.7: *Inserting the calibration values to your code*

```
1  const DeviceSettings deviceSettingsArray[] = {
2    // deviceName, servoZeroReferenceBytes, servo110ReferenceBytes,
       servoZeroReferenceDegrees, servo110ReferenceDegrees,
       servoZeroReferenceMicros, servo110ReferenceMicros,
       forceZeroReferenceValue
3    {"01_CLASSROOM_EDU_EXO_PRO", 499, 2478, 1800, 700, 2400, 1265, 1495, {
4        {499, 1800},   // 180.0
5        {671, 1700},   // 170.0
6        {855, 1600},   // 160.0
7        {1026, 1500},   // 150.0
8        {1212, 1400},  // 140.0
9        {1384, 1300},  // 130.0
10       {1562, 1200},  // 120.0
11       {1747, 1100},  // 110.0
12       {1925, 1000},  // 100.0
13       {2115, 900},   // 90.0
14       {2291, 800},   // 80.0
15       {2478, 700}    // 70.0
16     }
17   }
18 }
```

Hint

Note that we used 1800 instead of 180.0, this is because we wanted a higher precision while not using float values. Also note the naming of the variables *ZeroReferenceDegree*, equals 180 degrees, this is due to the servo motor orientation in the *Classroom Exo*. It is oriented in a way, that 180 degrees equals complete extension, which is our ”zero degree”.

Hint

The the used values are just for reference, they might be very different with your device!

5.7 Embedded Software

Summary

In this section we will walk you through the functionalities in the embedded code that is running directly on the SAMD21 chip of the Arduino Nano 33 IoT. With the embedded code, we enable Bluetooth Classic communication, different control algorithms for the motor, Inertial Measurement Unit (IMU) integration as well as battery and device status indications. This makes the *Classroom Exo* more User friendly, while also offering a compatible GUI in MATLAB to easily change between control modes.

5.7.1 Setting up Bluetooth Classic Communication

The Arduino Nano 33 IoT has two separate chips on its board. The SAMD21 ist the heart of the microcontroller, which is running the main program that handles serial communication, reading the sensors and controlling the servo motor. The second chip, the WiFi NINA 102 module, enables us to use WiFi, Bluetooth BLE or Bluetooth Classic to communicate with the world outside of the microcontroller. The WiFi Nina chip essentially repeats, what the bigger SAMD21 chip sent to it, and passes it through to the outside. This signal can then be received by our MATLAB GUI.

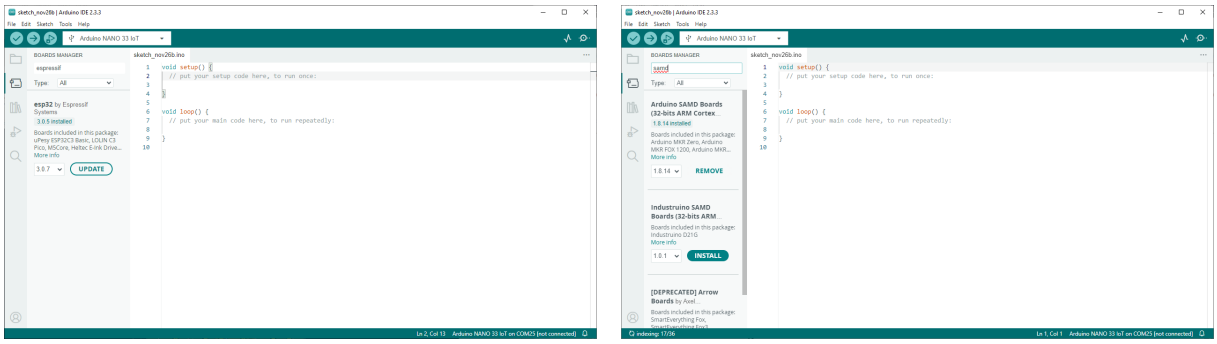


Figure 5.26: *Board Manager*

In order to get the Bluetooth communication going, you will need to do some tinkering with your board!

First, open the Arduino IDE, go to the Board Manager and install the following two Arduino cores:

- Arduino SAMD Boards (32-bits ARM Cortex-M3)
- Arduino ESP32 Boards

Next, open the Library Manager and download the following library into your IDE.

- WIFiNINA (by Arduino)

After installing the necessary libraries, we can now start to upload the code that will enable our board to communicate with Bluetooth.

1. Configure WiFi NINA 102 Board
   (a) Set the board to *Arduino NANO 33 IoT*
   (b) Open in sketch examples: File → Examples → WiFiNINA (Examples from Custom Libraries) → Tools → SerialNINAPassthrough
   (c) Upload *SerialNINAPassthrough.ino* to Arduino board
2. Set up Serial connection between SAMD & ESP32 cores:
   (a) Set board to ESP32 Dev Module
   (b) Change the following settings:
      i. Flash Mode: DIO
      ii. Flash Size: 2MB (16Mb)
      iii. Partition Scheme: Minimal (1.3 MB APP/700KB SPIFFS)
   (c) Open our *SerialtoSerialBT.ino* file
   (d) In line 6 change device_name to desired name (the Bluetooth device will show up with this name on your Windows device)
   (e) OPTIONAL: uncomment line 9 and change the PIN in line 10 to a more secure one
   (f) Upload *SerialtoSerialBT.ino* to Arduino board

In Listing 5.8 is a modified version of Arduinos *SerialToSerialBT.ino* example file. You can change the device name in line 6 to whatever name you prefer i.e. *01_CLASSROOM_EDU_EXO_PRO*. In the *setup()* function we send the device name to our main chip, so it knows what device it is, and can access device specific calibration values.

In the *loop()* we check if a Serial input is available from our main chip in line 23, and pass the available byte directly through to our Bluetooth output in line 24. The same happens in the other direction, we check if a byte is available in line 26 and pass it through to our main chip in line 27.

Listing 5.8: *Setting up Bluetooth Classic Operation*

```
1  #include "BluetoothSerial.h"
2
3  //#define USE_PIN // Uncomment this to use PIN during pairing. The pin is
       specified on the line below
4  const char *pin = "1234"; // Change this to more secure PIN.
5  String device_name = "xx_CLASSROOM_EDU_EXO_PRO";
6  BluetoothSerial SerialBT;
7
8  void setup() {
9    Serial.begin(115200);
10   SerialBT.begin(device_name); //Bluetooth device name
11
12   delay(500);
13
```

```
14    Serial.printf("The device with name \"%s\" is started.\nNow you can
          pair it with Bluetooth!\n", device_name.c_str());
15    #ifdef USE_PIN
16    SerialBT.setPin(pin);
17    Serial.println("Using PIN");
18    #endif
19  }
20
21  void loop() {
22
23    if (Serial.available()) {
24      SerialBT.write(Serial.read());
25    }
26    if (SerialBT.available()) {
27      Serial.write(SerialBT.read());
28    }
29  }
```

This *loop()* enables us to have a bi-directional Serial communication.

Now the classic Bluetooth is enabled on the Arduino Nano 33 IoT Board To program and use the Bluetooth device you can just write a normal Arduino code, change the board back to Arduino Nano 33 IoT and upload it from the IDE. In setup() initialize the device serial and Bluetooth serial communication as shown in Listing 5.9. Setting the *NINA_RESETN* pin to high is essential to activating the Bluetooth Antenna, so don't skip this step!

Listing 5.9: *Enabling Bluetooth Classic Operation in your main Arduino code*

```
1  setup() {
2    digitalWrite(NINA_RESETN;HIGH) //reset WiFi-Nina Board upon startup (
         connection to device is possible)
3    Serial.begin(15200); //serial comm from SAMD to ESP32
4    SerialNina.begin(15200); // serial comm from ESP32 to world (Bluetooth)
5     // here comes the rest of the code
6  }
```

Hint

There are two ways to test if you completed the Bluetooth Setup correctly:

- Check available Bluetooth devices on your mobile phone
- Check available Bluetooth devices in the Bluetooth settings on Windows

If the device shows up on both connection managers, you did everything right!

5.7.2 Testing the Bluetooth Communciation

Before we start with using the more complex control algorithms and our MATLAB GUI, we will check that the communication works in both directions.

Listing 5.10: *Testing Bluetooth Communciation*

```
1        connected_device = serialport("COM7",115200);
2        connect_answer = testConnection(connected_device);
```

For that upload the *Classroom_Exo.ino* file onto the Arduino Nano 33 IoT. Make sure to disconnect the Exo from your laptop afterwards and turn on the LED Power button. You should see the LED blinking in blue, with a frequency of 1Hz. This means, that the Exo is ready and waiting for a connection from Matlab. Next, we will open the Matlab file *BT_test_messaging.m*. Change the COM Port to the one your device is connected to and run the script, as seen in Listing 5.10.

You should see the confirmed connection in the Command Window, as well as the LED change from blinking to a solid color. This solid color can either be green, yellow, orange or red - depending on your battery percentage.

5.7.3 Messaging Protocol in the embedded Software

In order to communicate with the outside world, in our case the MATLAB GUI, we need to think about how we can create a fast and reliable messaging protocol. We want to send a command from our MATLAB GUI, such as changing the operation mode, reading the battery percentage or reading a specific sensor. For that we need a specific message/command so the micro controller understands, what to do next.

First, let's talk about our communication mode: we use a Serial communication both on our micro controller and on our receiving Windows device.

Of course, we could send a simple command like "getEMG" or "getBatVoltage" as a string from Matlab, and read that with the Serial.read() command on our micro controller. But, how does the micro controller now when it should listen for a command? How long will the message be? And how could we send settings back and forth?

We need an easily modifiable and adjustable communication, meaning we want messages of different lengths and purposes, but still adhering to the same principle. For that purpose we have developed a baseline message, that always looks the same, and can be extended to whatever we might need.

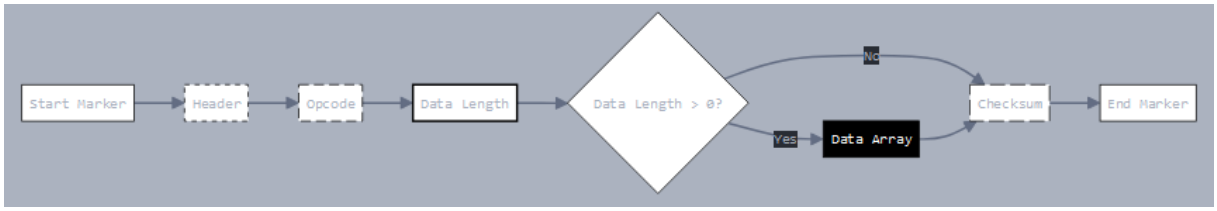


Figure 5.27: *Serial Communication Message Package Structure.*

As you can see in Figure 5.27 and Listing 5.11, our message starts with a header byte, which tells the micro controller to listen to whats coming next. Then we will send an opcode byte, telling the purpose

of the message. At the third index, there will be a datalength byte, telling the microcontroller how much data it should expect. The datalength byte in our baseline message is always zero. But if needed, can be adjusted to the amount of data, for example setting the motor speed. We close with a checksum, to ensure the packets integrity. Finally we have a terminator byte, that together with the datalength confirms that the message is now complete.

Listing 5.11: *Function for sending a new package via Bluetooth*

```
1  void sendMessage(message *msgOut) {
2    byte checksum = 0;
3
4    checksum = msgOut->header + msgOut->opcode + msgOut->dataLength;
5    msgOut->checksum = checksum;
6    uint8_t totalLength = 1 + 1 + 1 + 1 + msgOut->dataLength + 1 + 1;
7    byte message[totalLength];
8    uint8_t index = 0;
9
10   // Fill the byte array with the message components
11   message[index++] = startMarker;
12   message[index++] = msgOut->header;
13   message[index++] = msgOut->opcode;
14   message[index++] = msgOut->dataLength;
15
16   for (uint8_t i = 0; i < msgOut->dataLength; i++)
17   {
18     message[index++] = msgOut->data[i];
19   }
20
21   message[index++] = msgOut->checksum;
22   message[index++] = endMarker;
23
24   // Send the entire message with one Serial.write command
25   SerialNina.write(message, totalLength);
```

5.7.4 Sensor Data Acquisition and Filtering

As previously mentioned in our first guide, we will not be using delays in our embedded code, since the delay() command completely pauses the code execution. Instead we are using software timers with the *millis()* command.

Hint

If you have implemented a blinking LED from an Arduino before, you will have come across two ways of achieving that. Either with delay(), which blocks the code. Or with the non-blocking version using *millis()*. You can see how it works in the *BlinkWouthoutDelay.ino* example code in the Arduino IDE.

Listing 5.12: *Updating Sensor readings*

```
1  unsigned long currentTime = millis();
2
3  // Handle sensor updates
4  if (currentTime - previousSensorTime >= sensor_interval)
5  {
6    previousSensorTime = currentTime;
7    updateSensors(&mySensors);
8  }
```

First, we use the code in Listing 5.12. This code is like a timer that checks if it's time to read new sensor values. Imagine you have a stopwatch:

1. First, it looks at the current time
2. Then it checks if enough time has passed since the last sensor reading (sensor_interval)
3. If enough time has passed, it updates all the sensor readings and remembers when it did this

Think of it like checking your watch every few seconds to see if it's time to do something

The code in Listing 5.13 manages a special way of storing sensor readings using something called a "ring buffer." Here's what it does:

1. First, it reads new values from the two sEMG sensors, which measure muscle activity
2. It stores these new readings in a circular list (like a carousel that goes round and round)
3. It keeps track of the total sum of readings by:
   - Adding the new reading
   - Subtracting the oldest reading
4. Then it moves to the next spot in the circular list to prepare for the next reading

Listing 5.13: *Function for updating sensor readings in ring buffer*

```
1  void updateSensors(sensorData *data) {
2    // IMU data
3    ...
4
5    // EMG data
6    float emg0 = readEMGSensor(EMG0_SELECT_PIN);
7    float emg1 = readEMGSensor(EMG1_SELECT_PIN);
8    float newEmg[2] = {emg0, emg1};
9
10   for (int i = 0; i < 2; i++) {
11     data->emgSum[i] += newEmg[i] - data->emg[i][data->emgIndex];
12     data->emg[i][data->emgIndex] = newEmg[i];
13   }
14   // Move to the next index
15   data->emgIndex = (data->emgIndex + 1) % EMG_WINDOW_SIZE;
16
17   // Force data
18   ...
19
20   // Angle Sensor
21   ...
22 }
23 calculateMovingAverage(&mySensors);
24 }
```

It's like having a notebook with a fixed number of pages - when you reach the end, you go back to the first page and write over the old stuff!

Listing 5.14: *Calculating moving average of sensor readings*

```
1
2  void calculateMovingAverage(sensorData *data) {
3    // IMU data
4    ...
5
6    for (int i = 0; i < 2; i++) {
7      data->emgAvg[i] = data->emgSum[i] / EMG_WINDOW_SIZE;
8    }
9
10   // Force data
11   ...
12 }
```

This code in Listing 5.14 calculates the average of recent sensor readings. It's like finding the typical value from a group of measurements:

1. Takes the sum of recent readings
2. Divides by how many readings were taken

Think of it like calculating your average test score - add up all your scores and divide by how many tests you took! This averaging helps smooth out the sensor readings, making them more reliable by reducing random fluctuations. It's like looking at the general trend rather than each individual measurement.

Hint

For ease of reading, we have omitted the exact code for reading and storing the IMU, Force and Servo motor sensor values! But the functionality for those sensors is essentially the same as for the sEMG sensors! Keep in mind that you can find the entire code on our GitHub.

5.7.5 Control Algorithms in the embedded Software

Our embedded software enables you to run the *Classroom Exo* with different control modes, using the signals from the angle, force and sEMG sensor.

Embedded PID Controller

Imagine you're trying to turn your *Classroom Exos* arm to exactly 45 degrees. A PID controller helps you do this smoothly and accurately, like having a really careful and patient person controlling the movement. In order to first design the correct controller values, you'll need to know the transfer function model for your *Classroom Exo*. For this we used a small step response test. Using the recorded data and MATLABs *System Identification Toolbox* we approximated a transfer function model for the servo motor.

P = Proportional Control
Think of this like your immediate reaction:

- If you want to be at 45 degrees but you're at 30 degrees, you need to move 15 degrees
- The further you are from 45 degrees, the stronger the motor will turn
- It's like pushing harder on a door the more closed it is

I = Integral Control
This is like keeping track of past mistakes:

- If the arm has been stuck at 43 degrees for a while, never quite reaching 45
- The integral part slowly adds more power over time
- It's like getting gradually more determined to move something that's being stubborn

D = Derivative Control
Think of this like having brakes:

- If the arm is moving too quickly toward 45 degrees, D control slows it down
- It prevents overshooting (going past 45 degrees)
- It's like slowly easing off the gas pedal as you approach a stop sign

With our Graphical User Interface (GUI) you can design your PID controller using either the standard parameters like K_p, K_i and K_d, or you can use design and control parameters.

Thresholding Control using Force or sEMG Sensor

The Force Sensor, which is located at the wrist cuff, and the sEMG Sensor enable us to implement simple threshold based control algorithms. Namely the following three algorithms were implemented with the *Classroom Exo*:

The **biThresholdControl** function, in Listing 5.15, controls movement using two thresholds (like having two boundaries) using only one sensor.

Listing 5.15: ***biThresholdControl****: Uses two thresholds with one sensor*

```
1  void biThresholdControl(float sensorAverage, float upper_thresh, float
       lower_thresh) {
2    if (sensorAverage < lower_thresh)
3    {
4      positionDesired = min(positionDesired + angleStep/speed_setting,
         minAngle);
5    }
6    else if (sensorAverage > upper_thresh)
7    {
8      positionDesired = max(positionDesired - angleStep/speed_setting,
         maxAngle);
9    }
10
11   ...
12 }
```

Think of it like controlling the temperature in your house:

- You have a lower temperature (lower_thresh) and an upper temperature (upper_thresh)
- If it gets too cold (below lower_thresh), the position increases
- If it gets too hot (above upper_thresh), the position decreases
- If it's between these temperatures, nothing changes

The movement isn't instant - it's divided by a speed setting to make it smoother. It's like gradually adjusting a thermostat rather than making sudden changes.

The **biThresholdControl** function, in Listing 5.16, is simpler - it uses just one threshold (like having a single boundary).

Listing 5.16: ***monoThresholdcontrol****: Uses one threshold with one sensor*

```
1  void monoThresholdcontrol(float sensorAverage, float thresh) {
2    if (sensorAverage < thresh)
3    {
4      positionDesired = min(positionDesired + angleStep/speed_setting,
           minAngle);
5    }
6    else if (sensorAverage >= thresh)
7    {
8      positionDesired = max(positionDesired - angleStep/speed_setting,
           maxAngle);
9    }
10
11   ...
12 }
```

Imagine a light switch that works based on how bright it is:

- You have one brightness level you care about (thresh)
- If it's darker than this level (below thresh), the position increases
- If it's brighter than or equal to this level (above or at thresh), the position decreases

It's like an automatic light that turns on when it gets dark and off when it gets bright.

The third function, **monoThresholdBicontrol** in Listing 5.17 is special because it uses two different sensors but only one threshold.

Listing 5.17: ***monoThresholdBicontrol****: Uses one threshold with two sensors*

```
1  void monoThresholdBicontrol(float sensor1Average, float sensor2Average,
       float thresh) {
2    if (sensor1Average < thresh)
3    {
4      positionDesired = min(positionDesired + angleStep/speed_setting,
           minAngle);
5    }
6    else if (sensor2Average >= thresh)
7    {
8      positionDesired = max(positionDesired - angleStep/speed_setting,
           maxAngle);
9    }
10
```

```
11    ...
12 }
```

Think of it like controlling a door with two motion sensors:

- You have one sensitivity level (thresh) that applies to both sensors
- If the first sensor detects less motion than the threshold, the position increases
- If the second sensor detects more motion than the threshold, the position decreases
- If neither condition is met, the position stays the same

It's like having two different buttons that can control the same door, but each button works in a different way.

You can think of these three functions like different types of automatic controls - like how you might have different ways to control automatic doors (motion sensor, pressure plate, or both) but they all end up doing the same basic job of opening and closing the door. In this case their basic job is to use the force and sEMG sensor data to control up and down movement of the servomotor!

5.8 GUI in MATLAB

Summary

This section shows the MATLAB Front-End, which consists of an intuitive Graphical User Interface (GUI). The GUI communicates via Bluetooth Classic with te embedded code on the Arduino. The GUI enables controlling the *Classroom Exo* with a simulated P-/I-/D-Controller, Thresholding based on the Force or sEMG sensor. The wireless communication enables updating setting, streaming adds safety and freedom to move in space The fast communication also enables a simultaneous 3D animation of the *Classroom Exo* based on realtime IMU and angle data in the user interface. Additional information, like control mode and battery percentage, are also shown to the user.

5.8.1 Pairing your *Classroom Exo* with Windows

Hint

Pairing the Bluetooth device was only tested on a Windows 10 system. So we cannot guarantee that it works with other operating systems!

To use the device, you must pair it with your Laptop/PC:

1. Power the BT device on
2. In Windows go to System Settings → Bluetooth and other devices → Add devices → Bluetooth
3. Now search for new available devices and pair the desired device e.g. *xx_CLASSROOM_EDU_EXO_PRO*.

After pairing the device, you can use it with the MATLAB GUI.

5.8.2 Getting your MATLAB ready

Before we can use the GUI, you will need to install MATLAB and some additional Packages. Got to mathworks.com to create a Student Account and then download MATLAB. While installing we recommend to install the following packages, since they are essential for using the complete GUI:

- Control System Toolbox (version 23.2)
- Instrument Control Toolbox (version 23.2)
- Robotics System Toolbox (version 23.2)
- Sensor Fusion and Tracking Toolbox (version 23.2)
- Signal Processing Toolbox (version 23.2)
- Symbolic Math Toolbox (version 23.2)

Hint

The MATLAB GUI was developed and tested in Version R2023b, so we cannot guarantee that all the functionalities of the GUI will work with either older or newer versions!

5.8.3 Pairing your *Classroom Exo* with Windows 10 onward

Hint

Pairing the Bluetooth device was tested on Windows 10 and 11. So we cannot guarantee that it works with other operating systems! As of right now there is no support for MacOSx or Linux, and they are **NOT** planned for future releases!

To use the device, you must pair it with your Laptop/PC:

1. Power the BT device on
2. In Windows go to System Settings → Bluetooth and other devices → Add devices → Bluetooth
3. Now search for new available devices and pair the desired device e.g. *xx_CLASSROOM_EDU_EXO_PRO*.

After pairing the device, you can use it with the MATLAB GUI.

5.8.4 Getting started with the GUI

Open the MATLAB application by double-clicking on the file. Now, the MATLAB app will search for available Bluetooth devices, as you can see in Figure 5.28, that match our device naming structure.

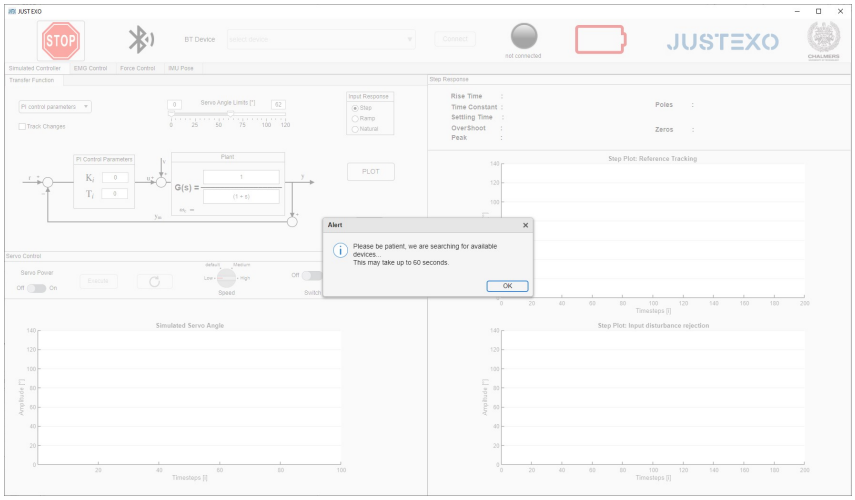


Figure 5.28: *Startup page of the MATLAB GUI searching for available Bluetooth devices.*

From the drop-down menu you are now able to select your desired device, see Figure 5.29, and just hit the *connect* button.

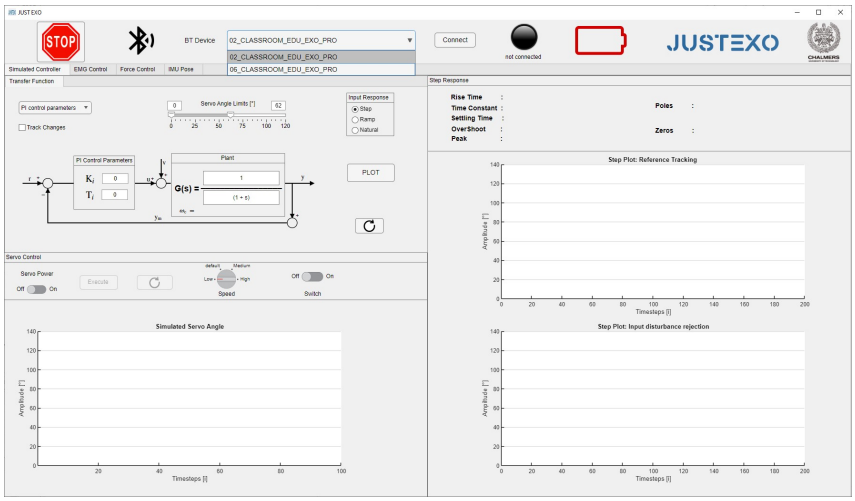


Figure 5.29: *Select your desired Classroom Exo device in the dropdown menu.*

When the connection was successful, you will be able to see the updated Bluetooth symbol, the status led and battery icon. For reference you can see the changed icons in Figure 5.30.

Figure 5.30: *Bluetooth connection with the Classroom Exo established.*

Now, you can select the tab with the control mode that you want to try out.

PID Control Tab

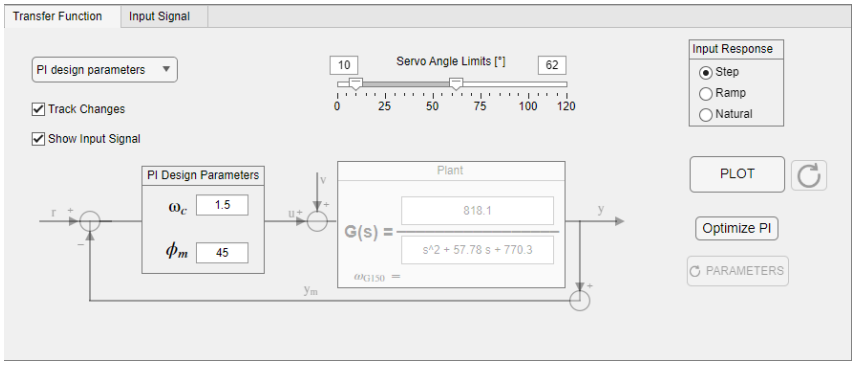


Figure 5.31: *The controller can be designed by different parameters, such as the design parameters ω_c and ϕ_m.*

With the initial values you will have a system response that exceeds safe limits. Now, you should adjust the values so you get the desired control system behaviour.

To speed things up a bit, we've implemented an optimization algorithm in the *Optimize PI* button. As you can see in Figure 5.32 the PI-controller responds a lot slower, and does not exceed the desired limits of the the step input signal.

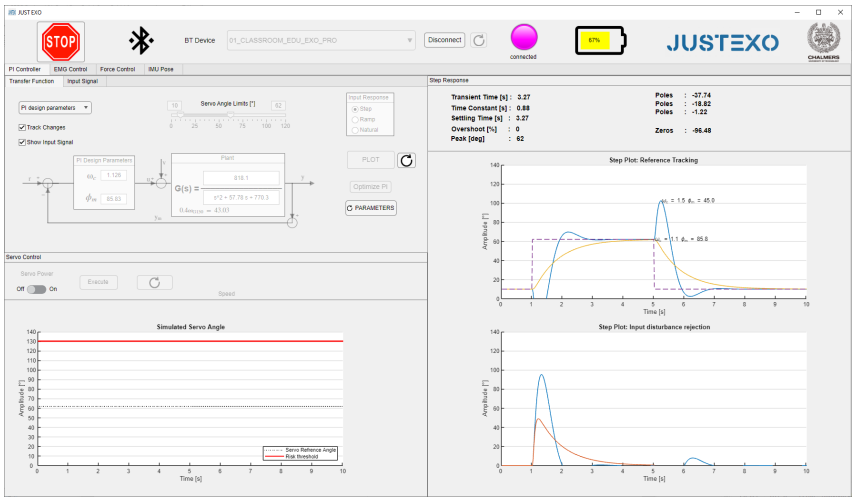


Figure 5.32: *The controller can be optimized by an automatic algorithm.*

Force Control Tab

If you want to use the threshold-based force control, go to the *Force Control Tab*. There a predefined settings for the thresholds and servo motor speed. You can just hit the *Start* button, and try out how the resistance feels against your wrist. You can see *Force Control Tab* interface in Figure 5.33.

You can adjust the force thresholds really easily: change the value on the threshold sliders and hit *Apply Thresholds* - this will restart the plot from zero with the newly selected threshold values.

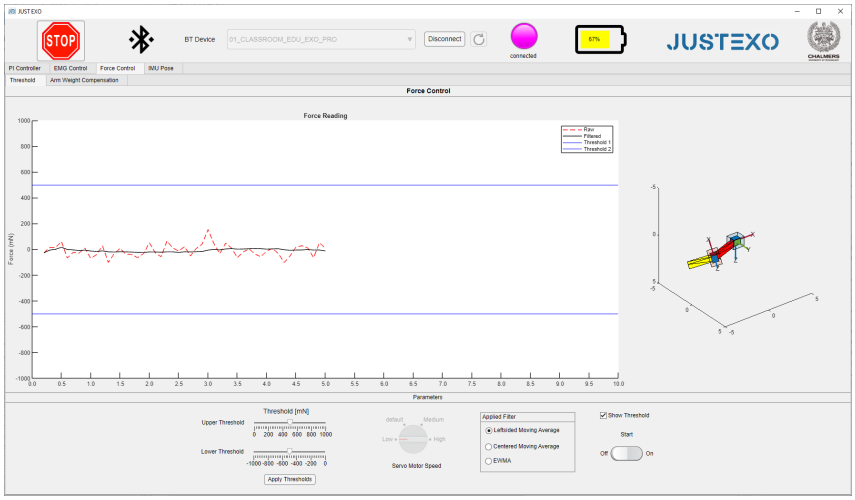


Figure 5.33: *The Force Control Tab interface enables you to control the servo motor by applying pressure to the wrist cuff on the Classroom Exo.*

You can also observe how different filters act on the signal quality. The filters on the radio-buttons are just for display purposes, and have no effect on the motor control itself.

On the right hand side, you can see a 3D-plot that displays a reference of the *Classroom Exo* based on IMU and angle sensor data. When you turn, or move the upper arm of your *Classroom Exo*, you can observe the movement in the liveplot. When the servomotor moves, you can see how the lower arm moves in reference to the upper arm.

sEMG Control Tab

If you now want to use the sEMG based control, go to the *EMG Control Tab*. You will find predefined settings for the thresholds and servo motor speed, as well as the sEMG channel. You can see *EMG Control Tab* interface in Figure 5.34.

Now, connect your sEMG sensor to a big muscle (i.e. biceps brachii) and plug the cable into the AUX port with the label *EMG 0*. If you want to use the other channel, you also have to change the channel in the drop-down menu!

Figure 5.34: *The sEMG Control Tab interface enables you to control the servo motor by flexing and relaxing your muscles.*

Again, you can hit the *Start* button and try to control the servo motor with your muscle activation.

On the right hand side, you will find the same 3D liveplot as in the *Force Control Tab*.

5.8.5 Disconnecting from your device

Before powering off your *Classroom Exo* it is crucial to disconnect your MATLAB GUI properly! Just hit the disconnect button, this will reset the *Classroom Exo* to it's initial mode - so you will see the led button blinking blue again. Your Matlab interface will reset, and start searching for paired devices again.

If you want to exit your GUI, you can now just close the MATLAB application; and also turn off the *Classroom Exo* manually.

5.8.6 Emergency Stop

If you do not feel comfortable using the *Classroom Exo* anymore, or a dangerous situation occurs there are two options of shutting down the servo motor movements.

1. Hitting the *Emergency Stop* button in the GUI - this turns off the motor power supply and disconnects the device from your MATLAB GUI. You will need to close the GUI before you can use it again (Figure 5.35).

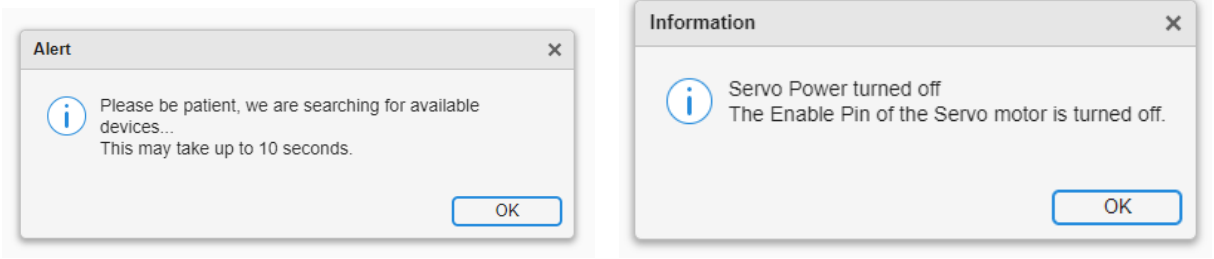
2. Hitting the LED power button on your *Classroom Exo* manually - this powers off the entire *Classroom Exo* without properly disconnecting from your MATLAB GUI. You will need to remove your paired device from the Windows settings, and start from the very beginning of this tutorial!

Figure 5.35: *The Emergency stop disabled the entire GUI and prompts you to shut off and restart the entire system.*

5.8.7 Error Messages and Troubleshooting

Alerts

The system alerts in the Figure 5.36a and Figure 5.36b are for your information only. They inform about the status during the initial Bluetooth device search, when the app is started. The second alert is for disabling the servo motor power - this enables the wearer to move their arm freely which is a lot more comfortable.

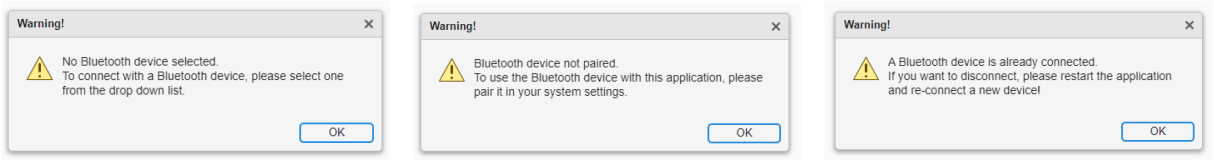


(a) *Alert that the device search is ongoing.* **(b)** *Alert that the servo motor power is turned off.*

Figure 5.36: *Examples of the two system alerts, these are for information.*

Warnings

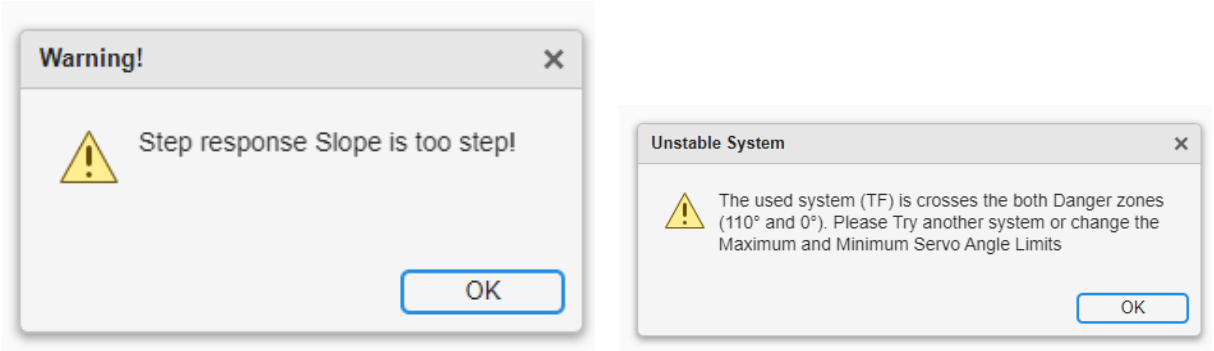
The warnings inform you if you did not select a Bluetooth devices before hitting the connect button (Figure 5.37a). If you did not pair your Bluetooth device within your Windows settings (Figure 5.37b). And last but not least if you have already connected your device (Figure 5.37c, in this case you have to restart the application. If the warning is still popping up, you will need to re-pair your device in your Windows settings.)



(a) **(b)** **(c)**

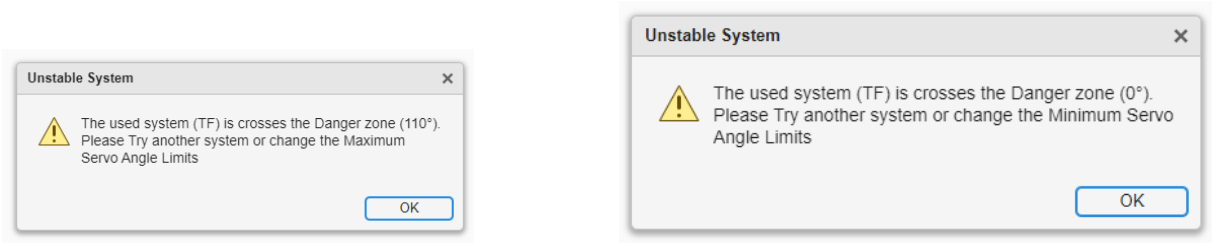
Figure 5.37: *Bluetooth connection related warnings.*

Next, when using the PID controller interface you wight encounter the following warnings in Figure 5.38a through Figure 5.39b.



(a) **(b)**

Figure 5.38: *Warnings related to the PID controller.*

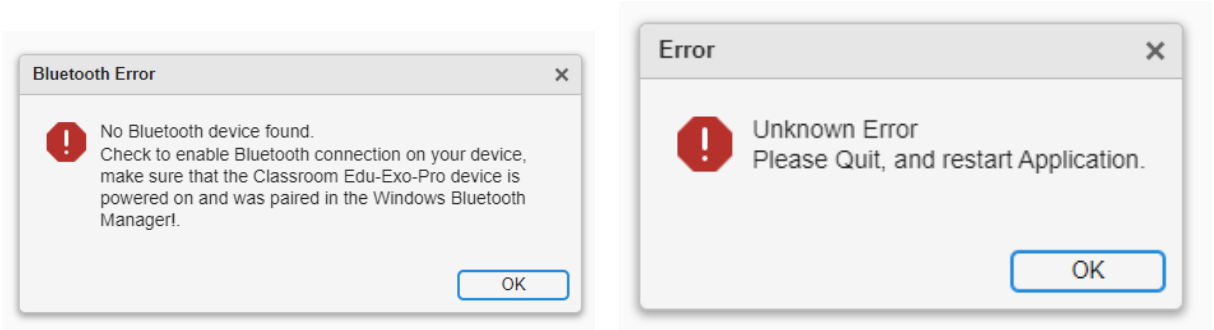


(a) **(b)**

Figure 5.39: *Warnings related to the PID controller.*

Errors

The message in Figure 5.40a lets you know that there was no Bluetooth device to be found. In this case go back to the instruction on how to pair your *Classroom Exo* on your Windows device! You have to close the MATLAB GUI before re-pairing the device.



(a) **(b)**

Figure 5.40: *Possible error messages.*

Figure 5.40b shows that the GUI encountered an unknown error, in this case disconnect from your device, close MATLAB and shut down the *Classroom Exo*. Let us know what you did while encountering that error, so we can try to fix it.

Acronyms

ABS Acrylonitrile Butadiene Styrene. 20
ADC Analog Digital Converter. 5

BOM Bill of Materials. i, v, 6, 7, 9

GUI Graphical User Interface. vi, 1, 36, 39, 44, 47, 48, 53, 55

HTML Hypertext Markup Language. v, 7, 8, 9

IC Integrated Circuit. 2, 9, 12, 13, 28
IDE Integrated Development Environment. vi, 27, 36, 38, 41
IMU Inertial Measurement Unit. 36, 43, 47, 51
IPA Isopropyl Alcohol. v, 8, 9, 18

LED Light Emitting Diode. vi, 2, 3, 9, 12, 14, 31, 53

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor. v, 2, 3, 10

PCB Printed Circuit Board. i, v, vi, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 18, 21, 22, 23, 24, 25, 27, 31
PETG Polyethylene Terephthalate Glycol. 20
PID Proportional-Derivative-Integral. vi, 44, 54, 55
PLA Polylactic Acid. 16, 17, 20
PPE Personal Protective Equipment. 8, 10, 18
PWM Pulse Width Modulation. 31

RGB Red-Green-Blue. 3, 14, 31
ROM Range Of Motion. 28

sEMG Surface Electromyography. i, v, 1, 2, 4, 5, 6, 13, 15, 18, 19, 24, 27, 30, 41, 43, 44, 46, 47, 52

SMD Surface-Mounted Device. v, 9, 10

SMT Surface Mount Technology. v, 9, 11, 14

USB Universal Serial Bus. v, 2, 9, 12, 13