# Final project

Yonatan Amir 207021858

Yuri Lukach 311809867

Index

[Final project 1](#_Toc156153649)

[Background 3](#_Toc156153650)

[Equipment 4](#_Toc156153651)

[Equipment Testing 14](#_Toc156153652)

[Planning 14](#_Toc156153653)

[Results 15](#_Toc156153654)

[Software 15](#_Toc156153655)

[Installation 15](#_Toc156153656)

[State machine 18](#_Toc156153657)

[Software Design 19](#_Toc156153658)

[M5Telemetry 19](#_Toc156153659)

[Block Diagram 20](#_Toc156153660)

[Schematics 20](#_Toc156153661)

[SharedDefines.h 21](#_Toc156153662)

[Running modes 21](#_Toc156153663)

[Slave 22](#_Toc156153664)

[Warnings 31](#_Toc156153665)

[Bugs 32](#_Toc156153666)

[Need to do 32](#_Toc156153667)

## Background

In this comprehensive document, we aim to provide a detailed overview of the array of sensors and components available for use in our ambitious project. Our primary objective is to establish an Arduino-based infrastructure that will serve as a foundation for future endeavors within the realm of Electrical Engineering, with a special focus on projects related to physiotherapy.

Our intention is to introduce and familiarize you with the diverse sensors that will play a pivotal role in this infrastructure. For each sensor, we will furnish you with not only general information but also valuable resources to aid in their utilization. These resources will include relevant links to tutorials, guides, and additional information sources, ensuring that you have access to a wealth of knowledge to effectively employ these sensors in your projects.

Furthermore, we will delve into the intricacies of conducting tests with each sensor, offering insights into their capabilities and potential applications. Equally important, we will outline how these sensors can seamlessly communicate and integrate with the various components that comprise our broader infrastructure, facilitating the creation of sophisticated and innovative projects within the field of physiotherapy and beyond.

By the end of this document, you will have a comprehensive understanding of the sensors at your disposal, the resources available to harness their capabilities, and the strategies to integrate them harmoniously into your Electrical Engineering projects, with a particular emphasis on their relevance to physiotherapy applications.

### Equipment

##### M5Stack

We will use MCU M5STACK CORES3 ESP32S3.



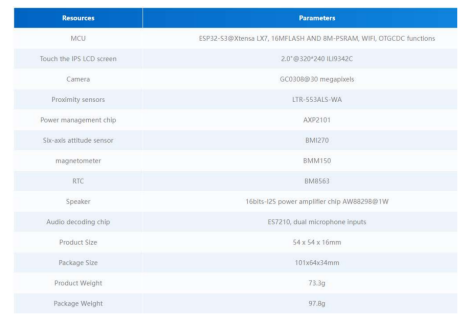
The primary rationale behind our choice to utilize the M5Stack as our controller for this project stems from several compelling factors. First and foremost, the M5Stack belongs to the ESP32 family of microcontrollers. This pedigree not only ensures robust performance but also provides compatibility with a wide range of software and hardware libraries, making it an ideal platform for our project.

Another noteworthy advantage of the M5Stack is its integrated sensor suite. Several sensors required for our project are already built into the M5Stack, greatly simplifying the wiring and setup process for students. This integrated approach not only enhances convenience but also reduces the potential for wiring errors, allowing students to focus more on the core aspects of their projects.

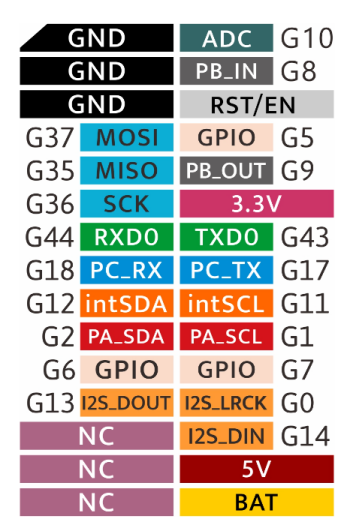
Additionally, the MCU (Microcontroller Unit) embedded within the M5Stack boasts a selection of materials that exhibit lower power consumption while delivering performance levels that are either on par with or even surpass the proposed alternatives. This efficiency is particularly advantageous for battery-powered applications, ensuring longer operating times and greater sustainability. Furthermore, the M5Stack can be seamlessly programmed using the familiar Arduino IDE, providing an accessible and user-friendly development environment for our students.

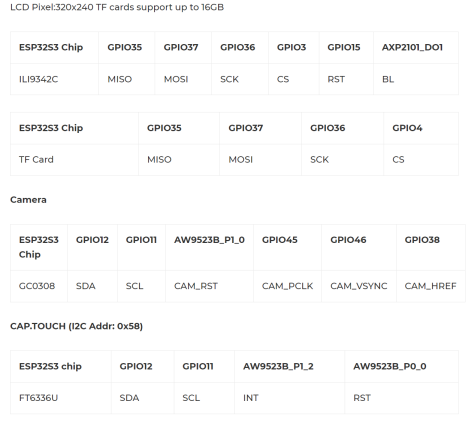
One of the standout features of the M5Stack is its flexibility in accommodating a variable number of sensors. Unlike more rigid systems that necessitate a fixed quantity of sensors, the M5Stack allows us to easily incorporate and adapt to a diverse array of sensors. This adaptability empowers students to explore a wide range of sensing capabilities and experiment with various sensor configurations, fostering creativity and innovation within their projects.

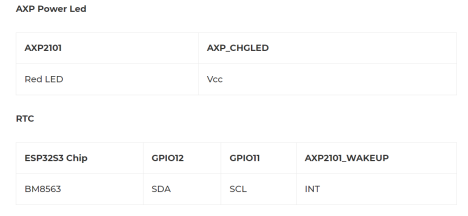
In summary, our selection of the M5Stack as the core controller for this project is underpinned by its ESP32 lineage, integrated sensors, energy-efficient materials, and Arduino IDE compatibility. Moreover, its flexibility in accommodating a variable number of sensors makes it an ideal choice, empowering students to explore and innovate in the dynamic field of sensor-based applications.



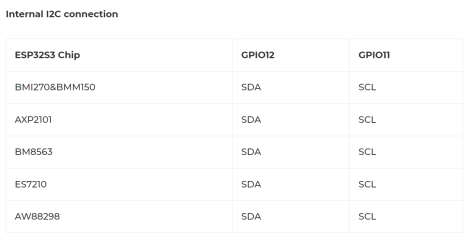
###### Pins











The relevant pins / ports will be:

* Port A – I2C
* Port B – GPIO,PWM and etc.
* USB.

##### PaHUB

The need to expand the I2C port is met with a HUB, through which we can add more sensors with a Grove connection. We use the AP9548PCA (B040-U), through which we can connect about 6 sensors.

Basic Specification:

The full specification of the AP9548PCA, on which the HUB is based, can be found.

https://m5stack.oss-cn-shenzhen.aliyuncs.com/resource/docs/datasheet/unit/pahub2/pca9548a. pdf

##### PbHUB

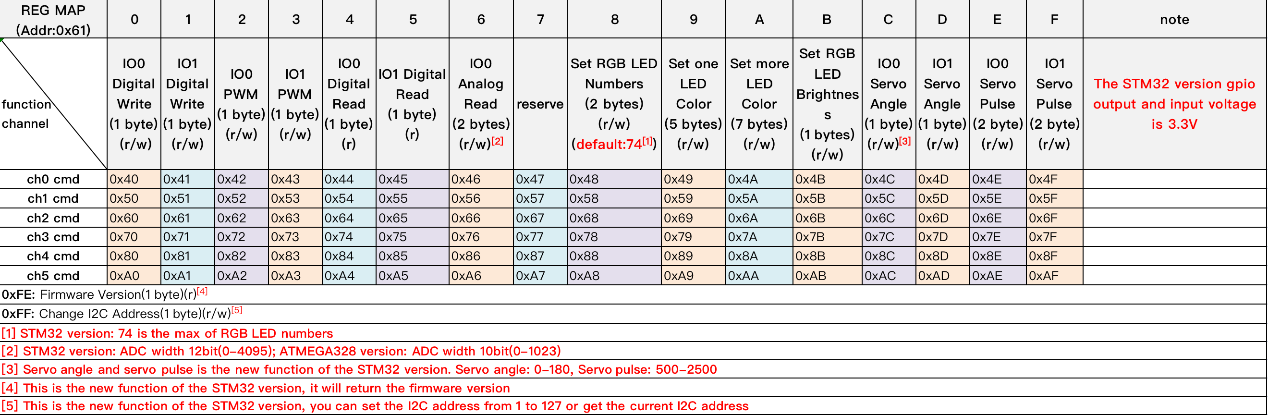
PbHUB Unit is a 6-channel expansion Unit with I2C control. Each Port B interface is capable of GPIO, PWM, Servo control, ADC sampling, RGB light control and other functions. Adopts STM32F030 for internal control.

**PbHUB will be connected to PaHUB.**

**https://docs.m5stack.com/en/unit/pbhub\_1.1**

###### Register map

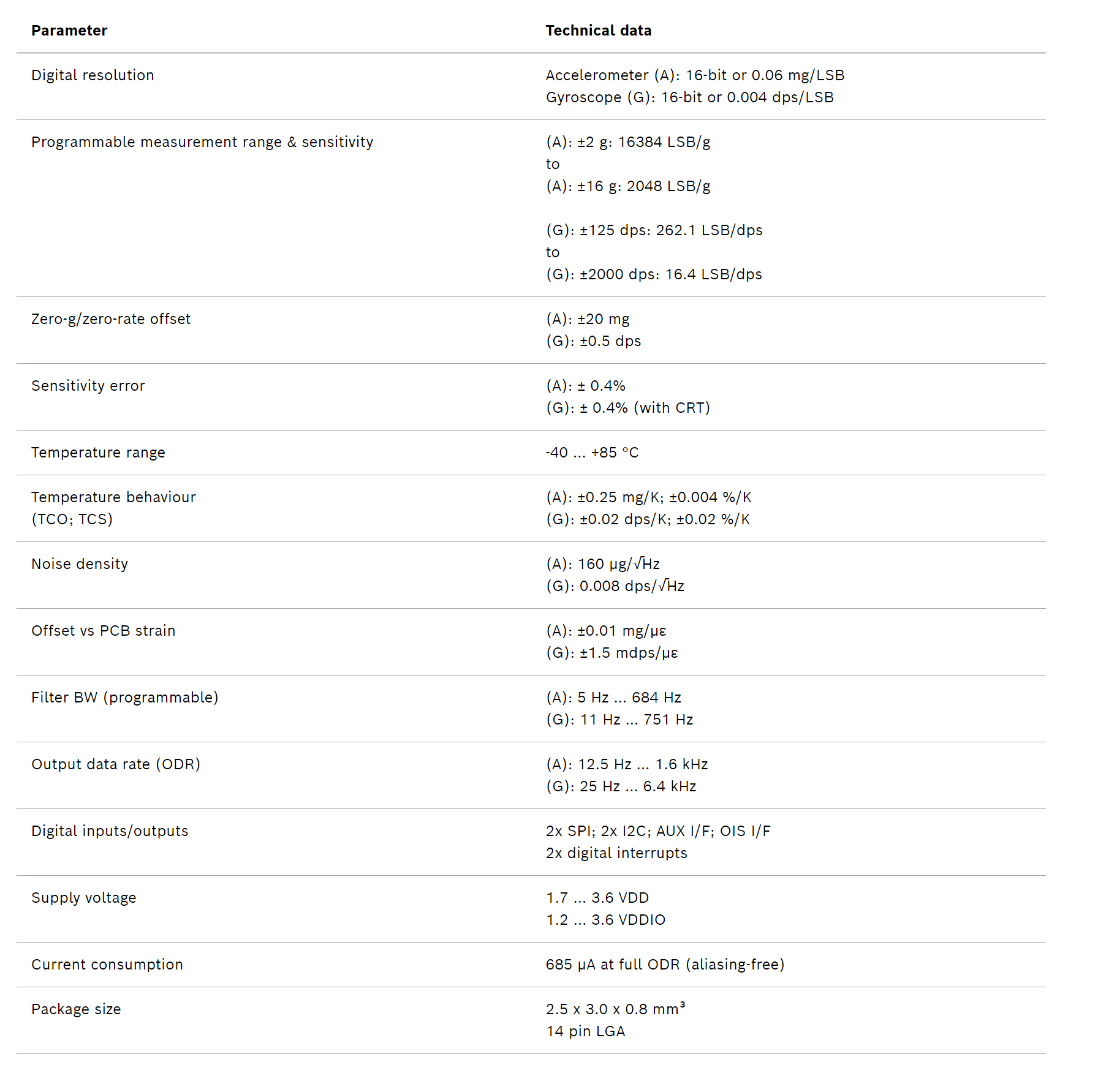
Register map for each port. To set/get register need to use i2c port.



##### Bosch BMI 270

The ultra-low power BMI270 is an IMU optimized for wearables providing precise acceleration, angular rate measurement and intelligent on-chip motion-triggered interrupt features.

The 6-axis sensor combines a 16-bit tri-axial gyroscope and a 16-bit tri-axial accelerometer featuring Bosch’s automotive-proven gyroscope technology.

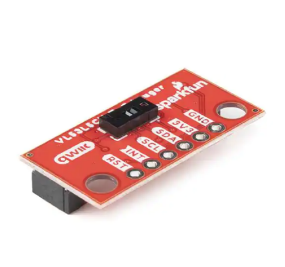


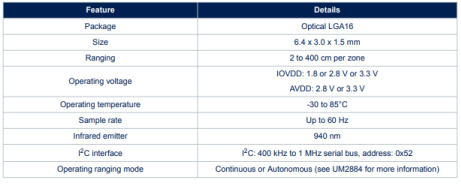
##### Proximity Sensor

The SparkFun Qwiic ToF Imager is a state of the art, 64 pixel Time-of-Flight (ToF) 4 meter ranging sensor built around the VL53L5CX from ST. This chip integrates a SPAD array, physical infrared filters, and diffractive optical elements (DOE) to achieve the best ranging performance in various ambient lighting conditions with a range of cover glass materials. Utilizing our handy Qwiic system, no soldering is required to connect it to the rest of your system. However, we still have broken out 0.1"-spaced pins in case you prefer to use a breadboard.

Multizone distance measurements up to 4000mm are possible across all 64 zones with a wide 63° diagonal field-of-view which can be read up to 15Hz. Thanks to ST Histogram patented algorithms, the VL53L5CX is able to detect different objects within the FoV. The Histogram also provides immunity to cover glass crosstalk beyond 60cm.

Ideal for 3D room mapping, obstacle detection for robotics, gesture recognition, IoT, laser-assisted autofocus, and AR/VR enhancement, the Qwiic connector on this sensor makes integration easy.





##### AMG8833

The SparkFun Grid-EYE Infrared Array Breakout board is an 8x8 thermopile array, meaning you have a square array of 64 pixels capable of independent temperature detection. It’s like having a thermal camera, just in a lower resolution. To make it even easier to to get your low-resolution infrared image, all communication is enacted exclusively via I2C, utilizing our handy Qwiic system. However, we still have broken out 0.1"-spaced pins in case you prefer to use a breadboard.

The on-board AMG8833 Grid-EYE from Panasonic possesses an accuracy rate of ±2.5°C (±4.5°F) with a temperature range of 0°C to 80°C (32°F to 176°F). Additionally, this IR "camera" board can detect human body heat at about 7 meters or less (that's about 23 feet), and has a frame rate of 10 frames a second to one frame a second. It is important to point out that while this version of the Grid-EYE is the high performance type with a high gain, it is only 3.3V tolerant.

[https://cdn.sparkfun.com/assets/4/1/c/0/1/Grid-EYE\_Datasheet.pdf?\_gl=1\*16butmf\*\_ga\*MTM3NjUxNTg3Ny4xNjg4NjMxOTEx\*\_ga\_T369JS7J9N\*MTY5NTU2Mzc5MS4xMy4xLjE2OTU1NjQwMTAuNjAuMC4w](https://cdn.sparkfun.com/assets/4/1/c/0/1/Grid-EYE_Datasheet.pdf?_gl=1*16butmf*_ga*MTM3NjUxNTg3Ny4xNjg4NjMxOTEx*_ga_T369JS7J9N*MTY5NTU2Mzc5MS4xMy4xLjE2OTU1NjQwMTAuNjAuMC4w)

##### FSR402

The FSR 402 model is a single-zone Force Sensing Resistor® optimized for use in human touch control of electronic devices such as automotive electronics, medical systems, and in industrial and robotics applications. FSRs are two-wire devices. They are robust polymer thick film (PTF) sensors that exhibit a decrease in resistance with increase in force applied to the surface of the sensor. Its active area is 14.7mm in diameter, and the sensor is available with four connection options. Interlink Electronics FSR 400 series is part of the single zone Force Sensing Resistor family.

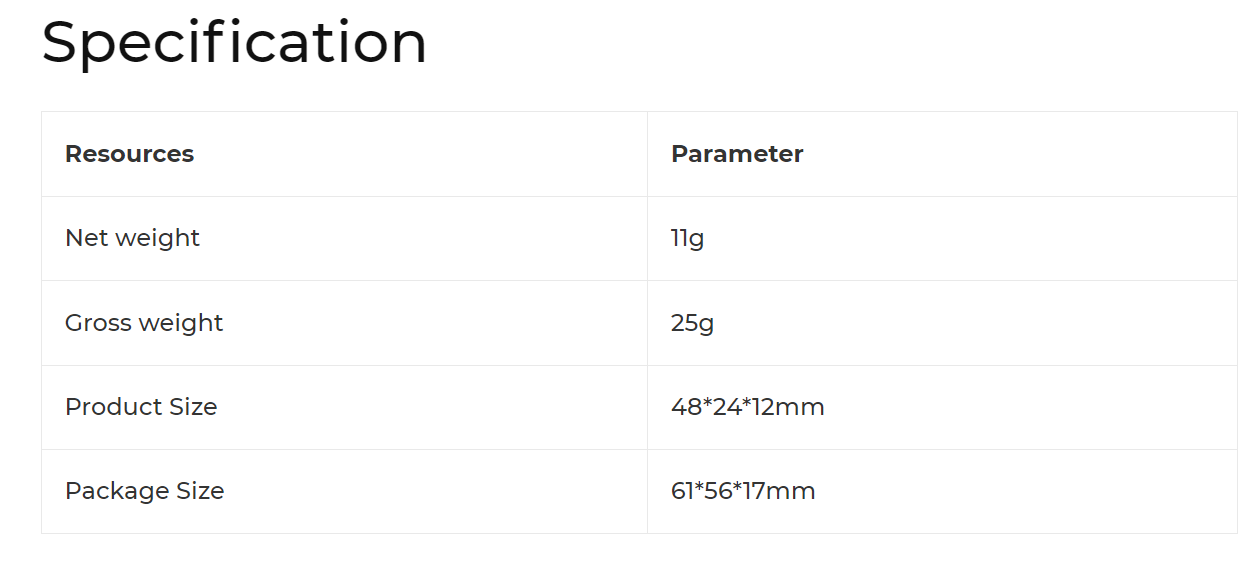
##### LCD

Model ILI9342C

##### Vibration Motor

Vibrator Motor is consisting of an N20 Motor and a metal eccentric wheel.

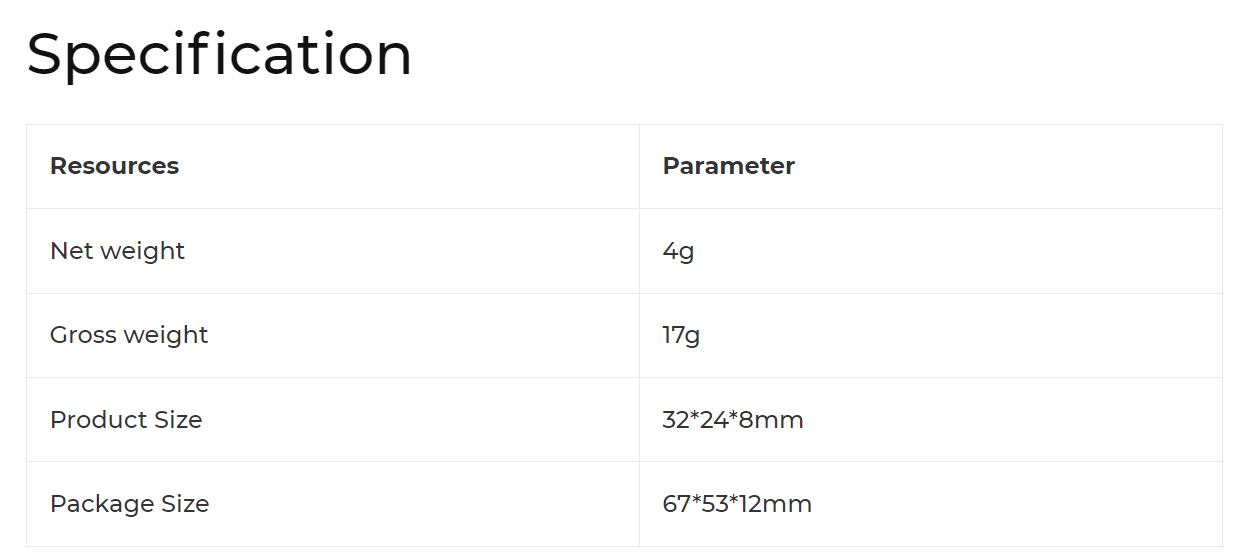
This N20 motor is has a 5V supply voltage. The output shaft has a rotational speed of 8800 RPM. Specifications can be seen below.



##### RGB

RGB is LED Unit include 3 individual LEDs. It is also one of the Unit from M5GO Kit. Each one can display any color based on the RGB spectrum. One feature of this Unit is that it's extendable, which means you can have multiple of them wired together.

This is a very useful piece of hardware for a STEM class, students can program it to realize some of cool applications, for example a traffic light.



## Equipment Testing

### Planning

**IMU (Inertial Measurement Unit)** - We will place the controller on various surfaces and read the telemetry of the IMU sensor. In each orientation, the IMU will provide a different vector inclined in only one positive or negative direction. Since we do not know the exact composition of the IMU, we will use this test to calibrate the IMU's axes relative to us.

**Proximity Sensor** - We will examine all 64 pixels of the sensor (8X8). For this purpose, we will place the sensor at the far end of an empty room and pass a test object in front of it at distances of 1 meter, 2 meters, and 4 meters. We will continuously query the sensor about the nearest distance and record the results. When the sensor responds to the test object, we will stop it. Additionally, we will test it at different angles and compare the readings at various measurement points against a commercial distance measurement device.

**Thermometer** - We want to check all 64 pixels of the sensor (8X8). We will position a test object so that it covers a single stack of pixels in height and query the temperature sensor. We will ensure that the returned temperature is within a reasonable tolerance for the human body. We will also pay attention to which pixels are returning temperature readings. The received distance can be checked using a proximity sensor or a distance measurement device.

**FSR (Force-Sensing Resistor)** - We will test this sensor by measuring different weights and comparing them to a known commercial weight.

**Vibration Motor** - We will ensure functionality by setting different DC levels and checking the motor's operation. Since the motor is an output component in the system, precise measurements (such as RPM) are less important to us than the vibration intensity.

**RGB (Red, Green, Blue)** - We will select colors with known RGB codes (not just the three primary colors) and ensure that we obtain the required color using their RGB values.

### Results

**TODO**

## Software

### Installation

#### Windows

##### Git

We recommend using github desktop which is user friendly GUI, but one can choose use git-bash directly.

* Github desktop download link: <https://desktop.github.com/>
* Git download link: <https://git-scm.com/download/win>

Repository link: <https://github.com/YonatanAmir1996/M5StackTelemetry>

Clone the following git: https://github.com/YonatanAmir1996/M5StackTelemetry.git

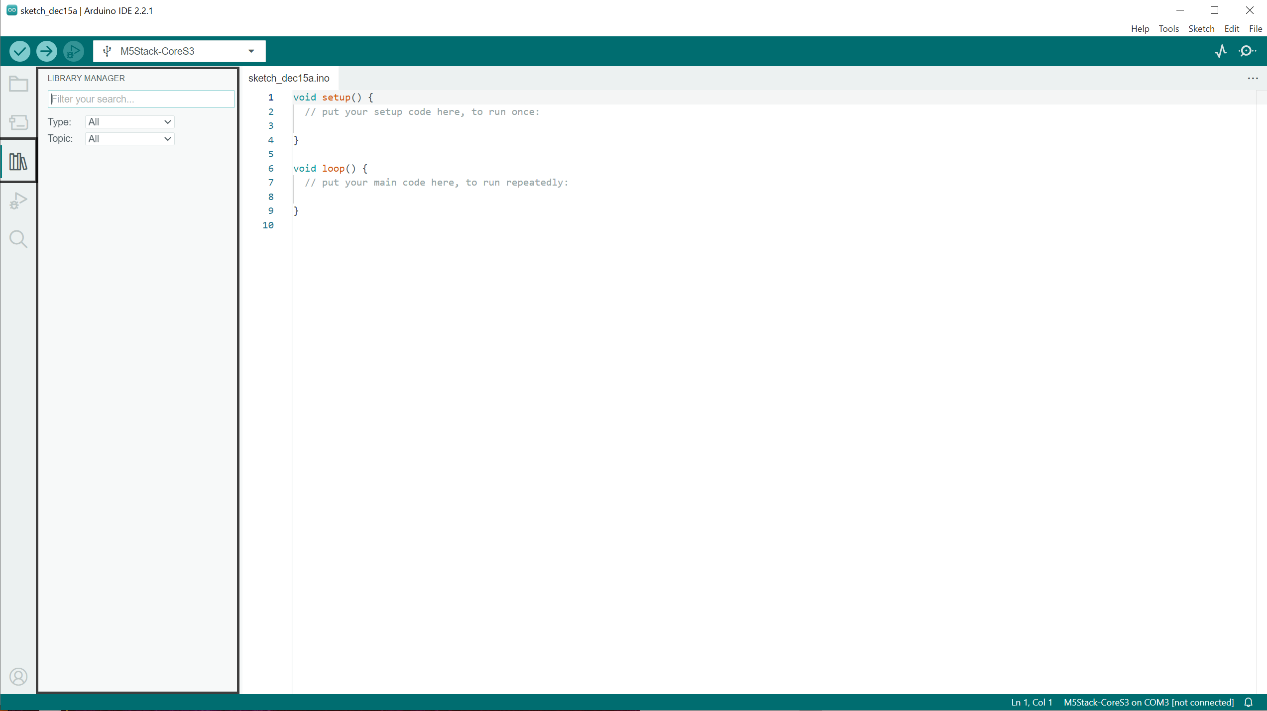
##### Arduino IDE

Download latest version: <https://www.arduino.cc/en/software>

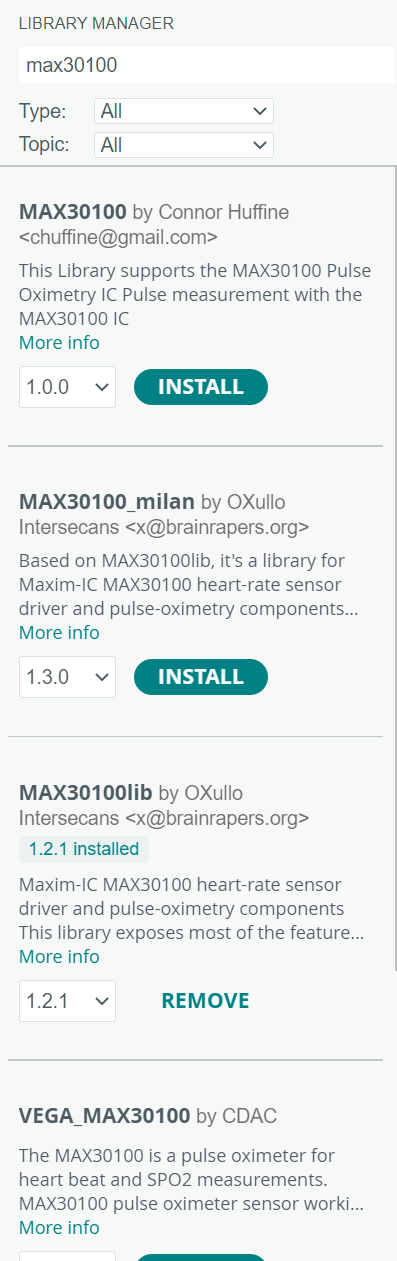
The link below is a tutorial installation provided by M5CORE team regard supporting M5Core in Arduino IDE:

<https://docs.m5stack.com/en/quick_start/m5core/arduino>

##### How to install libraries?



1. Click on libraries icon which located at right side of IDE, you'll notice that a small window which called library manager will appear.
2. Provide library name in filter your search and the following windows will popup:



3.Install the relevant library.

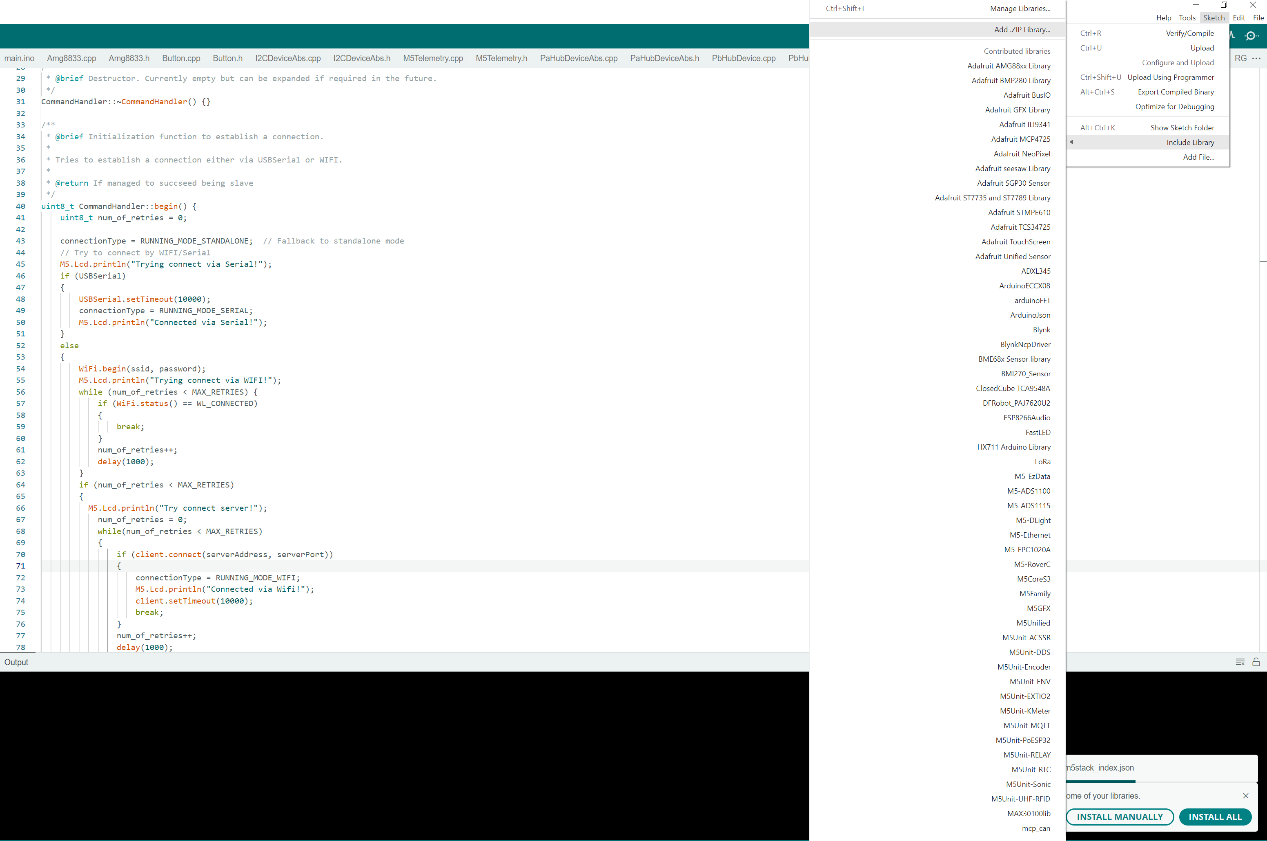
Libraries required:

1. Adafruit AMG88xx
2. Adafruit NeoPixel
3. SparkFun VL53L5CX Arduino

##### M5CoreS3 manual installation

In M5StackTelemetry repository there is a manual M5CoreS3 library. Need to install manually, follow the instructions

* Sketch->Include Library-> Add .ZIP library



* The M5CoreS3 API located at: <M5StackTelemetry>/Install. Note that <M5StackTelemetry> is the path to M5StackTelemetry directory.

#### Python 3.11

<https://www.python.org/downloads/release/python-3110/>

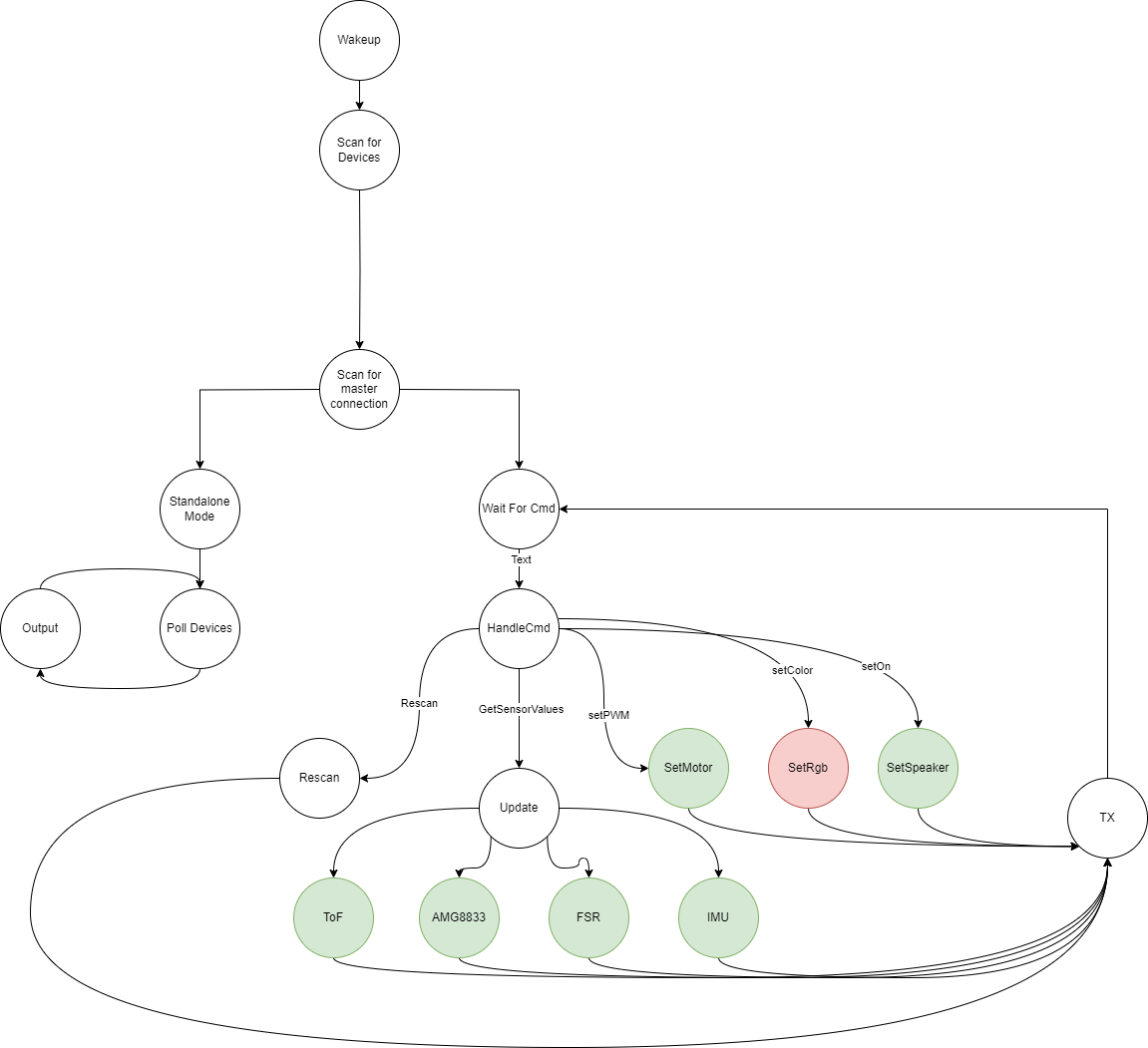
#### PyCharm

https://www.jetbrains.com/pycharm/

Libraries required:

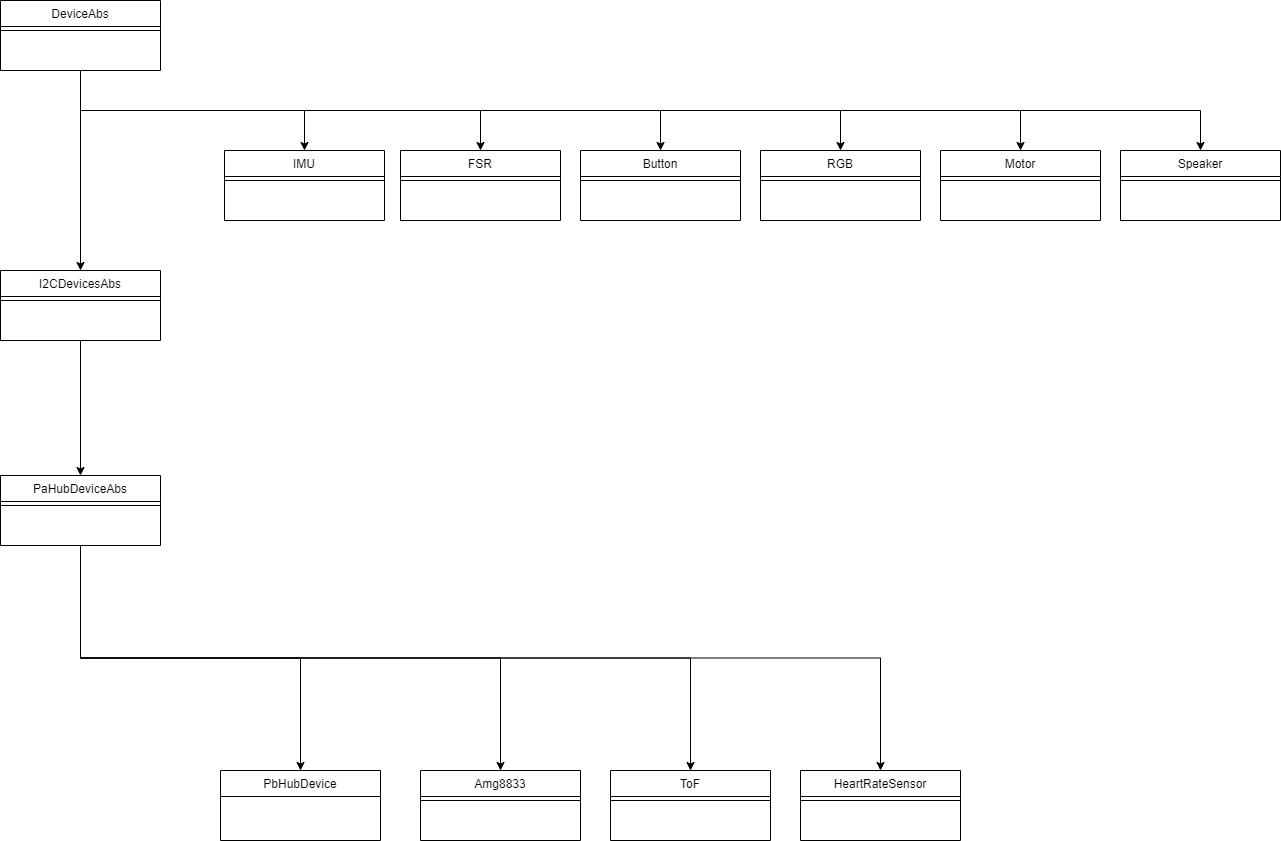
1. Pyserial
2. Numpy
3. Matplotlib
4. Pywin32
5. Scipy

## State machine



## Software Design

#### Device



## M5Telemetry

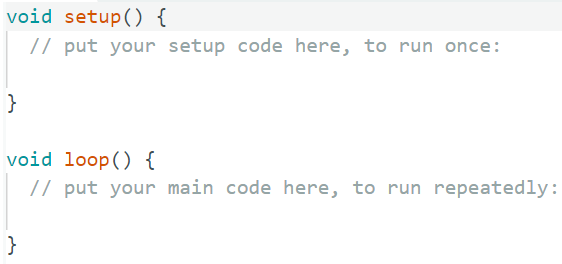
M5Telemetry is the main API which handles standalone / slave mode.

#### Arduino basics

#### Arduino

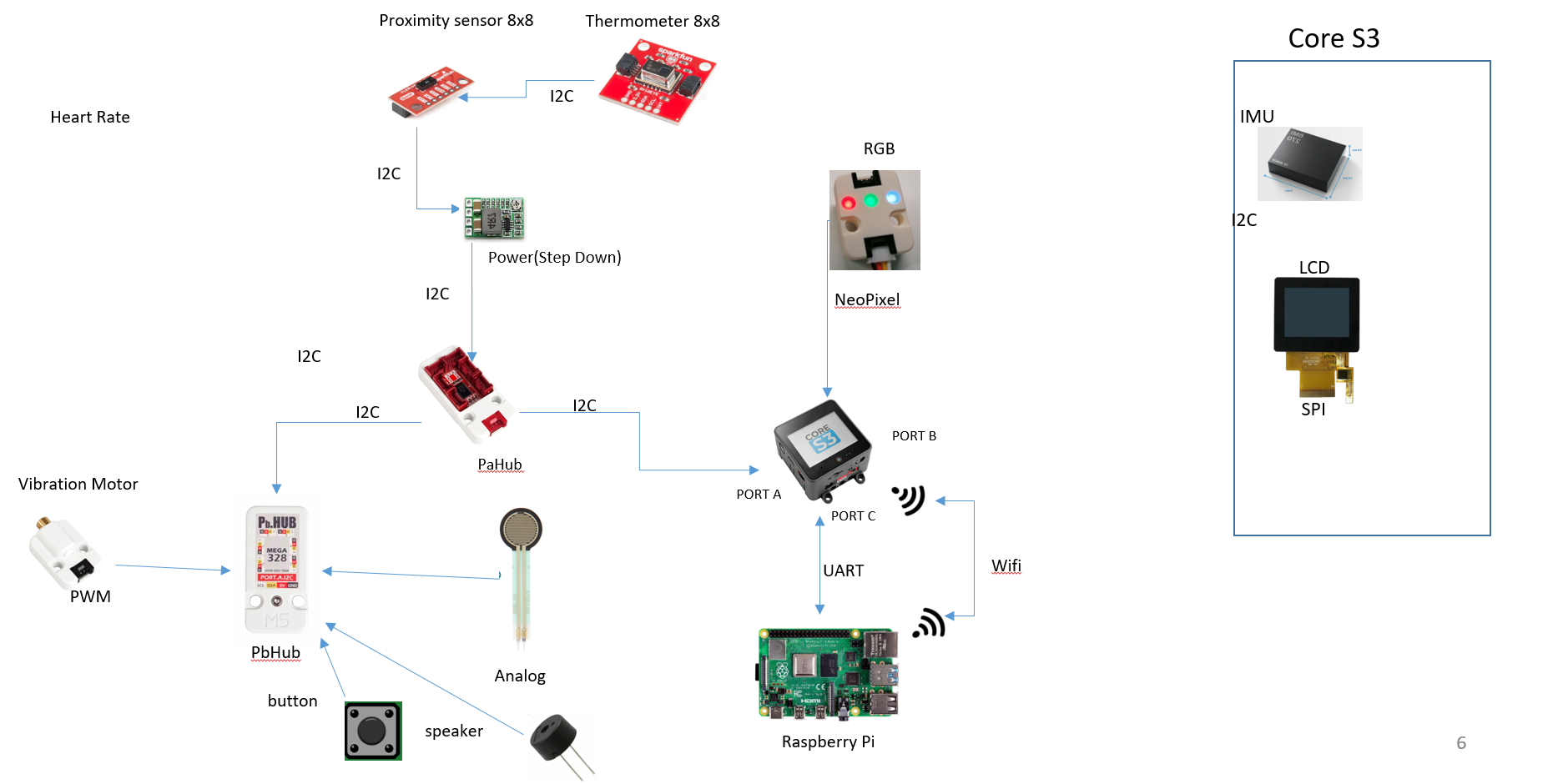
Arduino code is quite straightforward and typically consists of two main phases:

* Setup Phase: This phase runs once when the Arduino board is powered on or reset.
* Loop Phase: This is the main section of code that runs repeatedly as long as the Arduino is powered on.

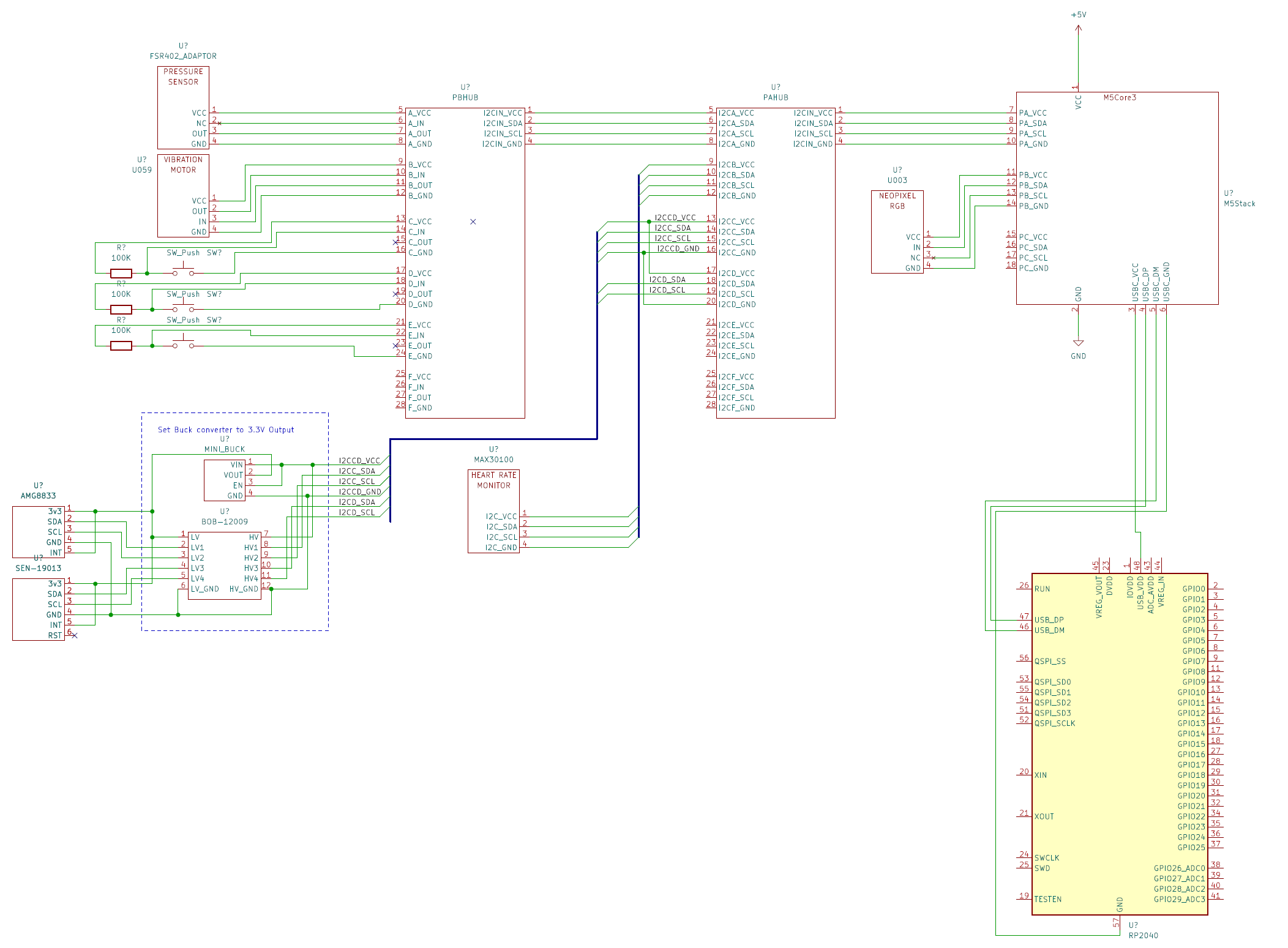


learning the basics of Arduino programming, you can review helpful tutorials on websites like [Tutorialspoint's Arduino section](https://www.tutorialspoint.com/arduino/index.htm).

### Block Diagram



### Schematics



### SharedDefines.h

This header file serves as a central repository for essential information, including I2C addresses, register assignments for each pin, enum declarations, and more. The enum declarations can also be found within SharedDefines.h.

## Running modes

#### Standalone

Standalone will run in 2 cases:

1. Force standalone run.
2. No master serial connection was found(WIFI/Serial).

Code:

#include "M5Telemetry.h"

#include "SharedDefines.h"

void setup()

{

    M5Tel.begin();

}

void loop()

{

    M5Tel.run(

        True,                     // Force standalone flag

        /\* Standalone parameters in case of force standalone / failure connect to RASPBERRY PI\*/

        PB\_HUB\_PORT\_0\_ADDR,       // Button PbHub address

        PB\_HUB\_PORT\_INVALID\_ADDR, // FSR PbHub address

        PB\_HUB\_PORT\_INVALID\_ADDR, // Vibration Motor PbHub address

        PB\_HUB\_PORT\_INVALID\_ADDR, // speaker Address

        false                     // use RGB device(Supported only in PORT B)

        );

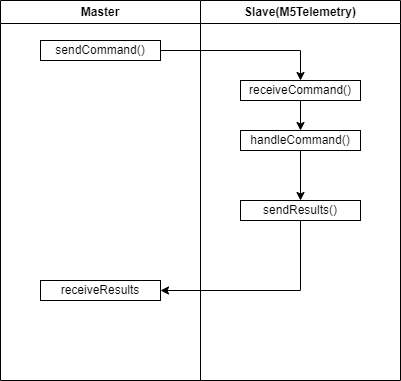
}

In standalone the after scanning devices – sensors values will be updated endlessly. To switch between output **using the button which MUST BE Connected via PbHUB!**

### Slave

An API which called in our code as 'CommandHandler' we can handle connection via Serial/Wifi in case of slave mode and sending buffer to server (M5Stack device is the **client**).

#### High level design

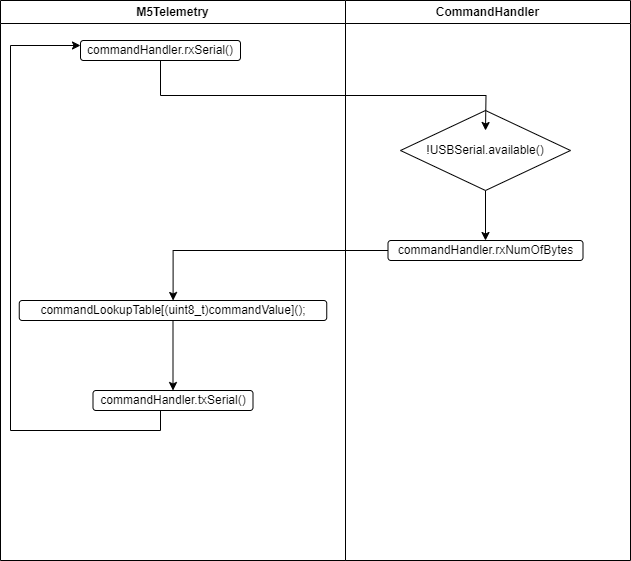


There are two buffers used for command handling:

1. **RxBuffer:** This buffer stores words (argument values) received from the master.
2. **TxBuffer:** This buffer contains raw data from the device. It is the responsibility of the user to parse the buffer, as exemplified in the case of COMMAND\_RUN\_SENSORS.

#### Serial

##### Diagram



##### Usage

**By default,** the M5 will try find a serial connection (via UART) at baud rate of 115200 – if been able to connected a CLI instance will be need to run in order to retrieve data from sensors.

Just ensure when connecting via serial connection that Arduino actually recognize it.

#include "M5Telemetry.h"

#include "SharedDefines.h"

void setup()

{

    M5Tel.begin();

}

void loop()

{

    M5Tel.run(

        false,                    // Force standalone flag

        /\* Standalone parameters in case of force standalone / failure connect to RASPBERRY PI\*/

        PB\_HUB\_PORT\_0\_ADDR,       // Button PbHub address

        PB\_HUB\_PORT\_INVALID\_ADDR, // FSR PbHub address

        PB\_HUB\_PORT\_INVALID\_ADDR, // Vibration Motor PbHub address

        PB\_HUB\_PORT\_INVALID\_ADDR, // speaker Address

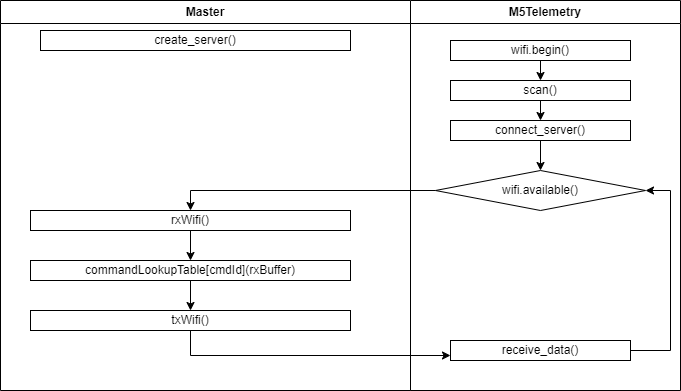
        false                     // use RGB device(Supported only in PORT B)

        );

}

#### WIFI

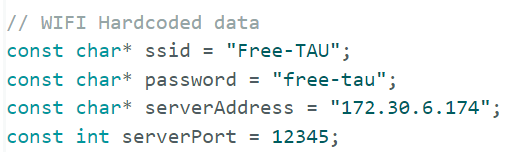
##### Diagram



##### Usage

To connect via WIFI, need to ensure **Serial** connection is closed, or prevent connect directly to PC / Raspberry PI.

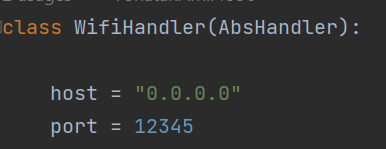
In **<M5StackTelemetry>/main/CommandHandler.c** there's the following section:



Those relevant field to connect via Wifi. Need to compile software in order to update such fields.

* ssid – Wifi name
* password – wifi password
* server address – python server address
* server port – server port

Python side <M5StackTelemetry>/CLI/Assets/WifiHandler.py

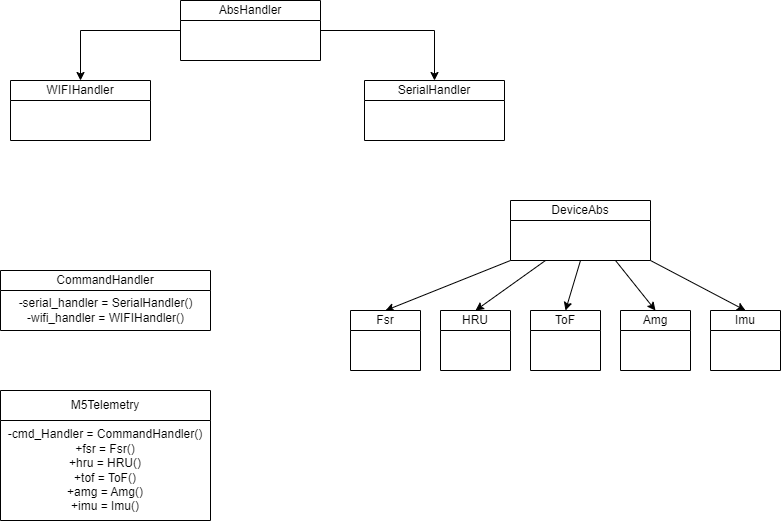


Host need to be aligned with serverAddress field in M5StackTelemetry FW code.

ServerPort need to be aligned with port of WifiHandler.

##### Python(Master)

###### Class hierarchy



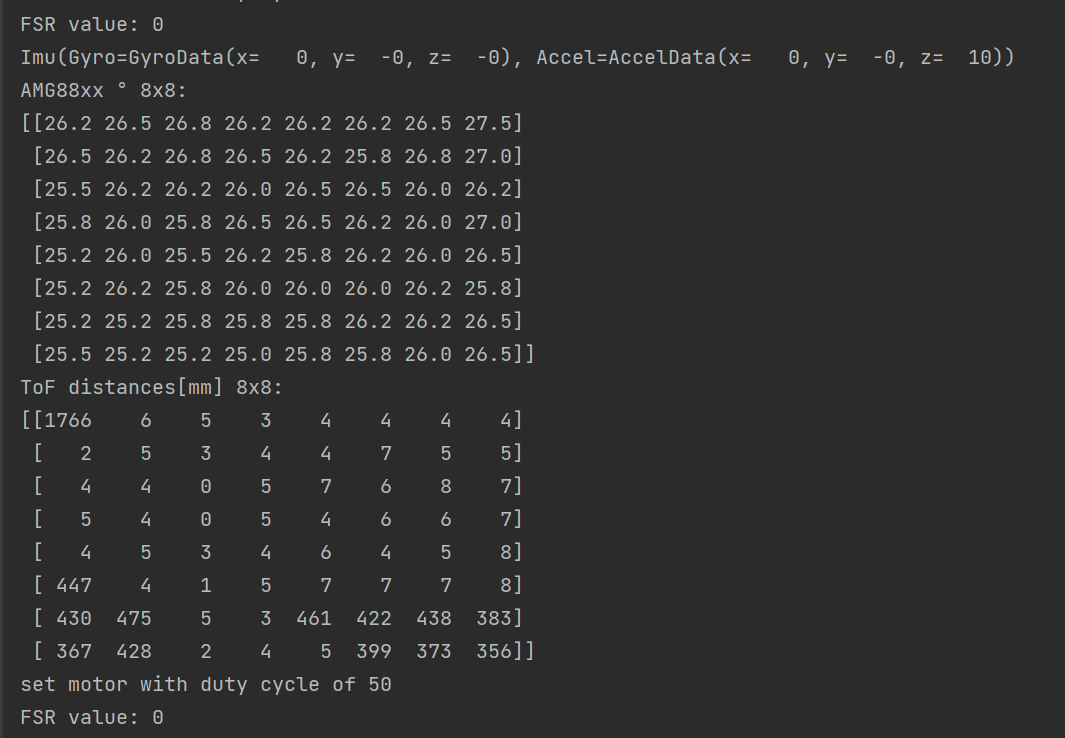
###### Usage example

By using PyCharm API - data from device may be achieved. Each command is created in **master** must be aligned with M5Telemetry **API!**

**Example** of built-in commands:

import os  
import sys  
import time  
  
root\_path = os.path.join(os.path.dirname(os.path.abspath(\_\_file\_\_)), "../../")  
sys.path.append(root\_path)  
  
# Import necessary classes and enums from respective modules  
from CLI.M5Telemetry import M5Telemetry  
from CLI.Devices.DeviceAbs import Device\_e  
from CLI.Assets.CommandHandler import PbHubPortAddr\_e  
  
  
def poll\_devices(m5\_telemetry\_interface: M5Telemetry, to\_set: bool):  
 *"""  
 This function polls various devices to retrieve and possibly set new data.  
 :param m5\_telemetry\_interface: The M5Telemetry object used for getting and setting data.  
 :param to\_set: A boolean that decides whether to set new data or not.  
 """* # Retrieves data from specified devices (TOF, IMU, FSR, AMG833)  
 m5\_telemetry\_interface.update\_values([Device\_e.TOF, Device\_e.IMU, Device\_e.FSR, Device\_e.AMG833])  
  
 # Print results from sensors. Users may use the results from devices for further operations.  
 print(m5\_telemetry\_interface.fsr) # Accessing the 'fsr' attribute from M5Telemetry object.  
 print(m5\_telemetry\_interface.imu) # Accessing the 'imu' attribute from M5Telemetry object.  
 print(m5\_telemetry\_interface.amg) # Accessing the 'amg' attribute from M5Telemetry object.  
 print(m5\_telemetry\_interface.tof) # Accessing the 'tof' attribute from M5Telemetry object.  
  
 if to\_set is True:  
 # Setting values on various devices through the M5Telemetry object.  
 m5\_telemetry\_interface.command\_set\_speaker()  
 m5\_telemetry\_interface.command\_set\_rgb(0, 100, 0, 0)  
 m5\_telemetry\_interface.command\_set\_motor(50)  
 else:  
 # Resetting RGB and motor values to default through the M5Telemetry object.  
 m5\_telemetry\_interface.command\_set\_rgb(0, 0, 0, 0)  
 m5\_telemetry\_interface.command\_set\_motor(0)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Creating an instance of the M5Telemetry class.  
 interface = M5Telemetry()  
  
 # Rescanning various devices to set their addresses.  
 interface.rescan(button\_pb\_hub\_addr=PbHubPortAddr\_e.PORT\_0,  
 fsr\_pb\_hub\_addr=PbHubPortAddr\_e.PORT\_1,  
 vibration\_motor\_pb\_hub\_addr=PbHubPortAddr\_e.PORT\_3,  
 speaker\_pb\_hub\_addr=PbHubPortAddr\_e.PORT\_5,  
 is\_rgb\_connected=True)  
 set\_output = True  
 while True:  
 # Continuously poll devices and possibly set new values based on the set\_output flag.  
 poll\_devices(interface, set\_output)  
 set\_output = not set\_output # Toggle the set\_output flag.  
 time.sleep(2) # Sleep for 2 seconds before polling the devices again.

The results will be:

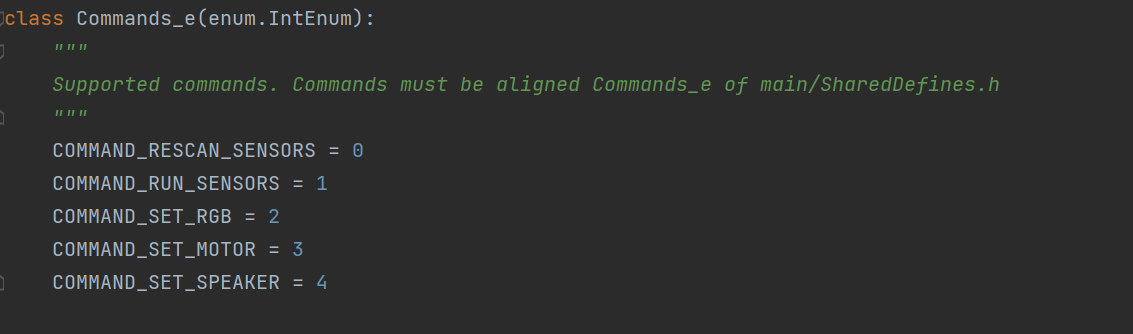


##### How to add new command

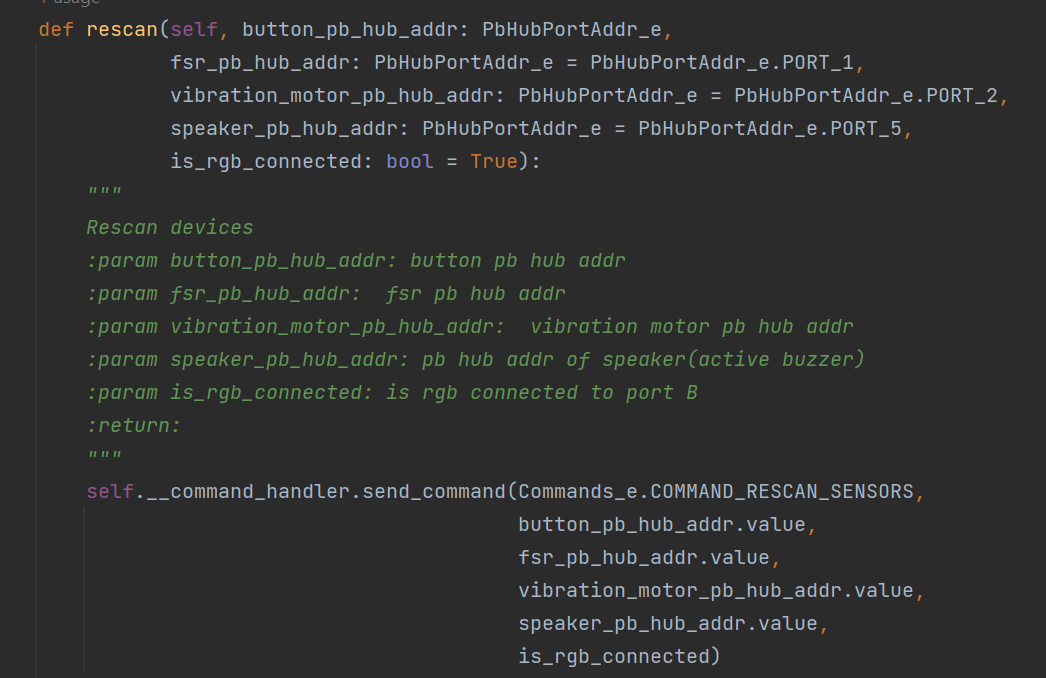
Add command name into 2 Commands\_e enum(master+slave). **Note that enums must be aligned in values otherwise it may lead to unexpected behavior!**

###### Command enum in python

Add command in Commands\_e: CLI/Assets/CommandHandler

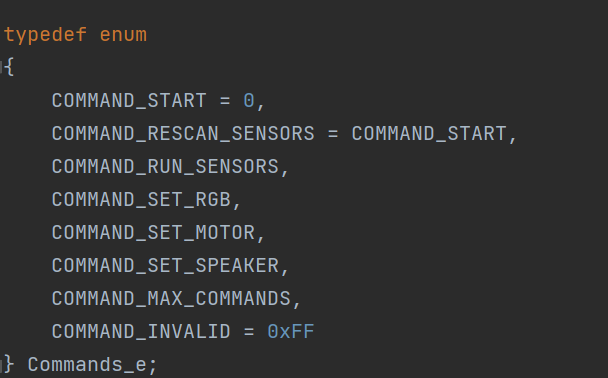


Afterwards create in M5Telemetry the command with specified arguments and pass to send\_command CommandHandler method the command name and the specified arguments:



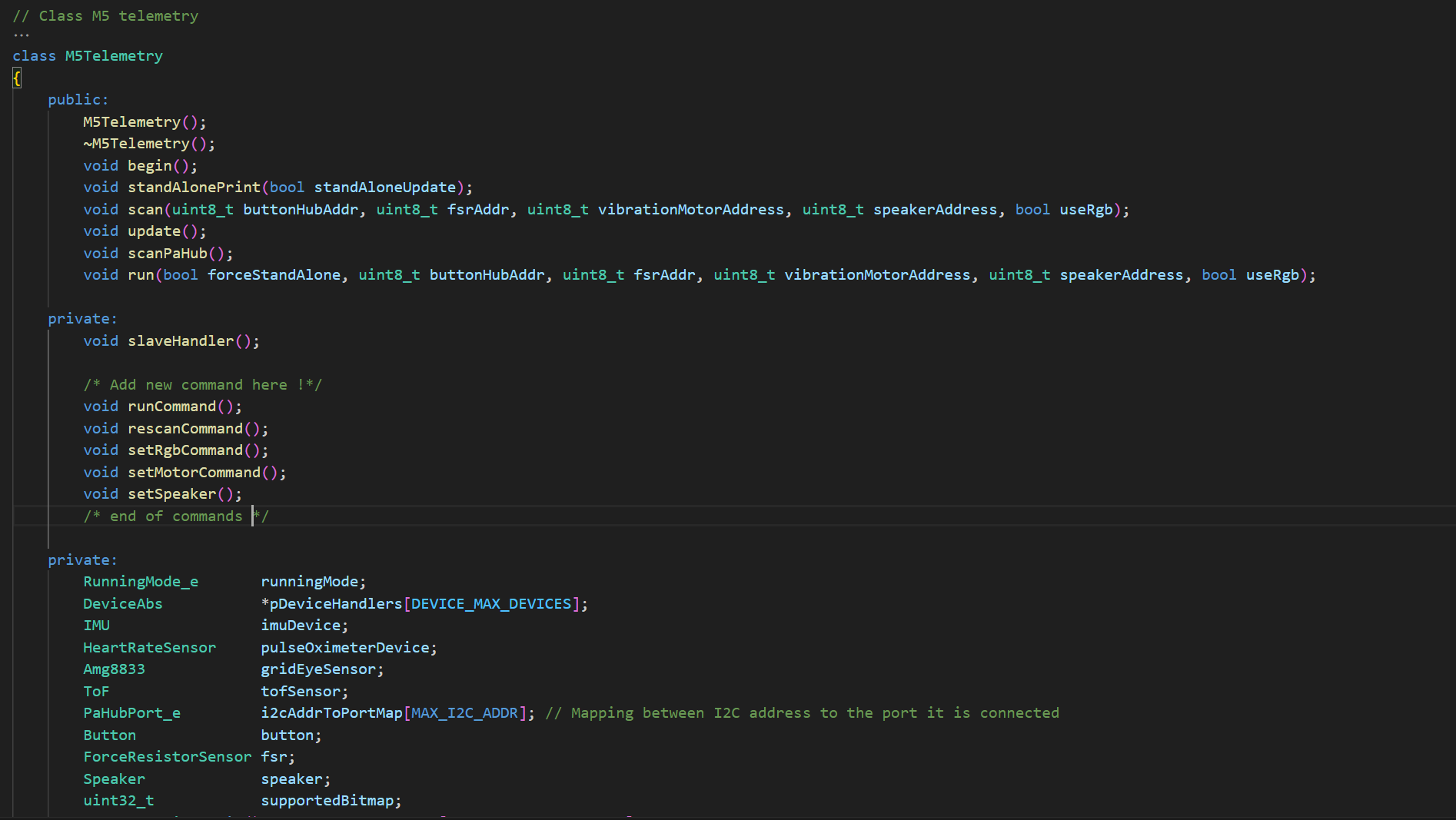
Rescan device example

Slave Commands\_e location: main/SharedDefines.h

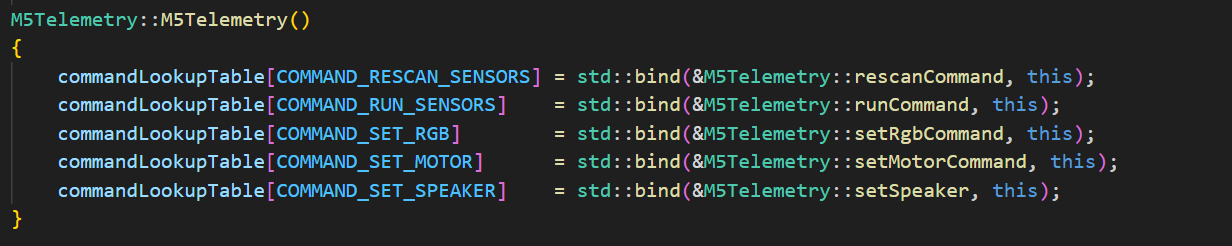


**Note that command must be before MAX\_COMMANDS!**

Create a method in M5Telemetry API between comments:



Add a callback function to lookup table in the constructor of M5Telemetry. (otherwise exception / unexpected behavior will raise), the main idea of the lookup table is for easier maintenance:



The arguments will be saved word-by-word from the 8th byte in buffer.



In case to return results to user – fill TxBuffer. For reference follow runCommand in M5Telemetry API.

# Warnings

* The AMG8833 and ToF sensors, which employ the I2C protocol, are connected in a serial configuration. Overloading this connection with too many devices might cause signal complications (pull downs).
* Initially, the I2C and qwiic cables (Relevant to ToF and AMG8833) were incorrectly connected due to misleading information found online. This was later rectified by manually resoldering and reversing the cables.

## Bugs

* RGB isn't lights up in M5StackTelemetry and example mode.

# Need to do

* Fix RGB issue
* Integration with Raspberry PI team
* Equipment testing - results
* Pictures
* Installation tutorial(Raspberry)
* **What is maximum possible frequency work with ToF? ( Roi req ).**