# Final project

Yonatan Amir 207021858

Yuri Lukach 311809867

Index

[Final project 1](#_Toc146467456)

[Background 3](#_Toc146467457)

[Equipment 3](#_Toc146467458)

[M5Stack 3](#_Toc146467459)

[PaHUB 7](#_Toc146467460)

[PbHUB 8](#_Toc146467461)

[Bosch BMI 270 9](#_Toc146467462)

[Proximity Sensor 9](#_Toc146467463)

[AMG8833 10](#_Toc146467464)

[FSR402 11](#_Toc146467465)

[LCD 11](#_Toc146467466)

[Vibration Motor 11](#_Toc146467467)

[RGB 12](#_Toc146467468)

[Software 12](#_Toc146467469)

[Arduino IDE 12](#_Toc146467470)

[Python 3.11 12](#_Toc146467471)

[PyCharm 12](#_Toc146467472)

## Background

In this document, we will present information about the sensors and components that we offer to use in our project - building an Arduino-based infrastructure for future projects in Electrical Engineering with an emphasis on projects in the field of physiotherapy. We will introduce the sensors with which we will assist. For each sensor, the general information about it will be written, relevant links on how to use it (tutorials and additional information) will appear, the relevant test, how to communicate with the rest of the infrastructure components.

## Equipment

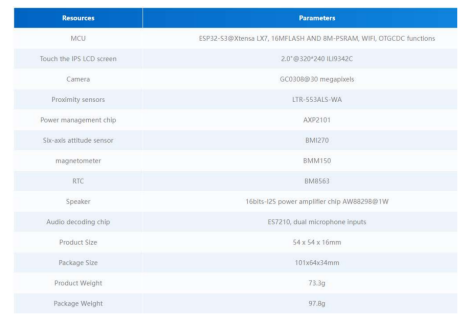
### M5Stack

We will use MCU M5STACK CORES3 ESP32S3.

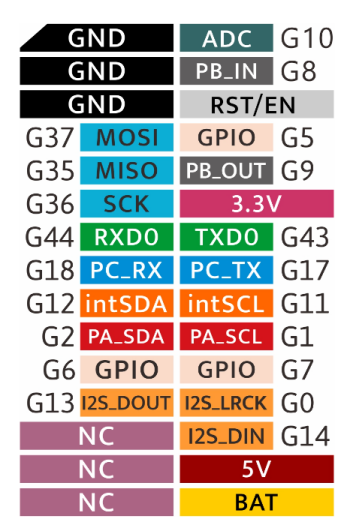


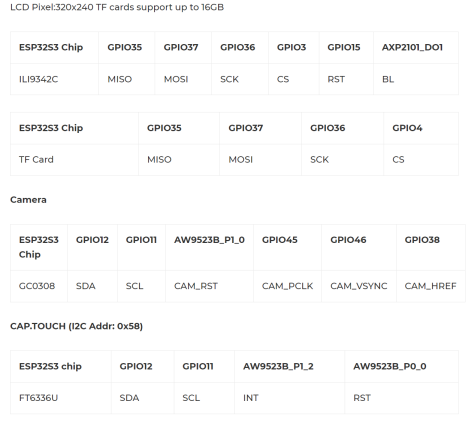
The main reasons for using M5Stack are the fact that it is a controller from the ESP32 family, and in addition, some of the sensors required for the project are built into it, so it will be able to facilitate the wiring for the students. Furthermore, some of the materials built in the MCU are materials with a lower power need and almost the same/better level of operation than what was proposed. It is possible to develop in the Arduino IDE work environment.

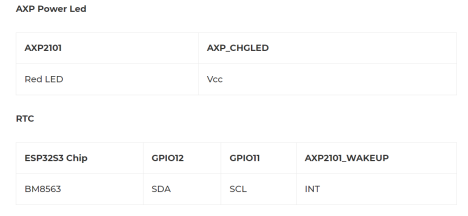
Working with M5Stack will allow us to work with a variable amount of sensors, unlike a more static system that would require a fixed amount of sensors.



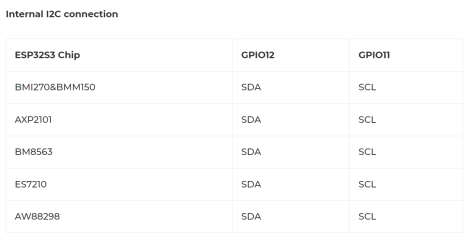
#### Pins











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

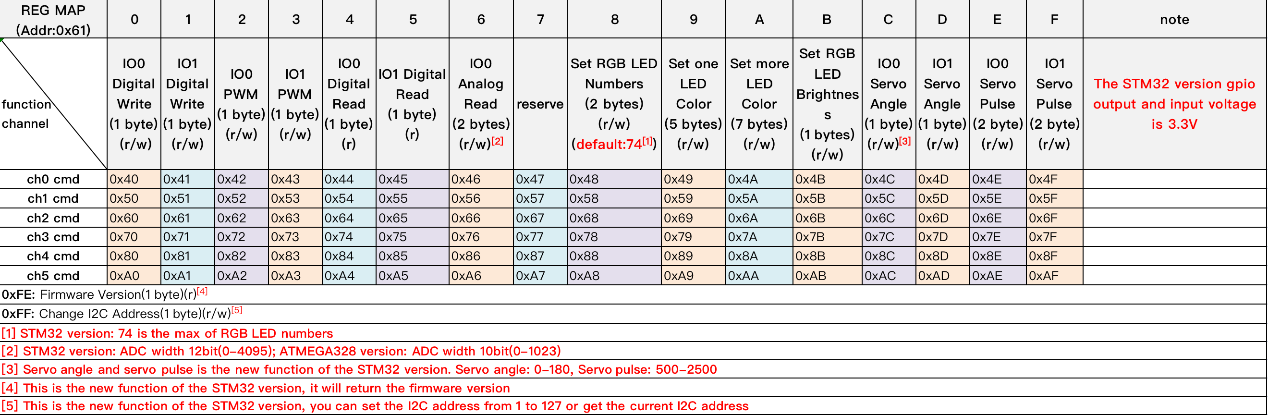
PbHUB Unit is a 6-channel expansion Unit with I2C control. Each Port B interface is capable of GPIO, PWM, Servo control, ADC sampling, RGB light control and other functions. Adopts STM32F030 for internal control.

**PbHUB will be connected to PaHUB.**

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

#### Register map

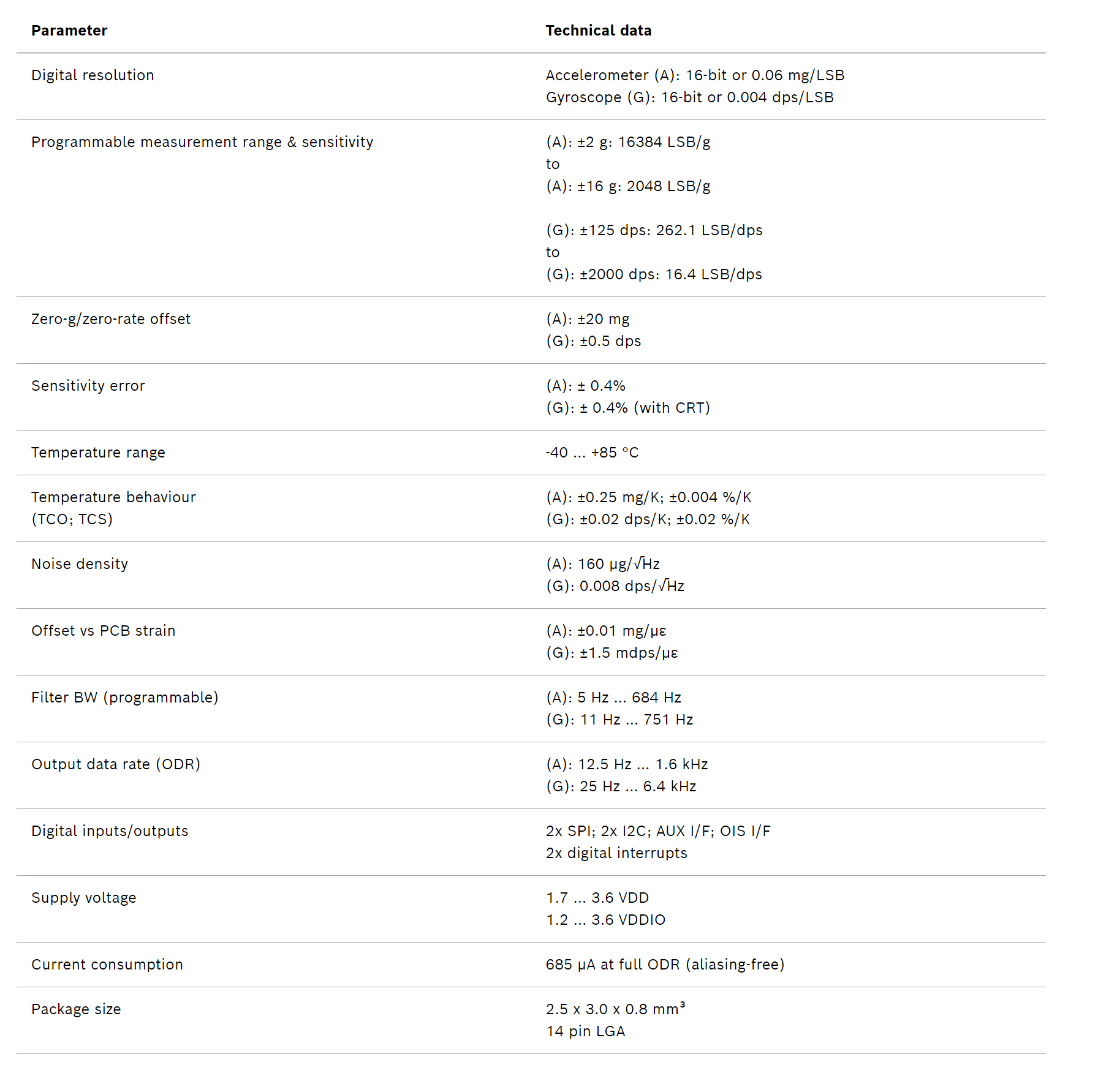
Register map for each port. To set/get register need to use i2c port.



### 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.

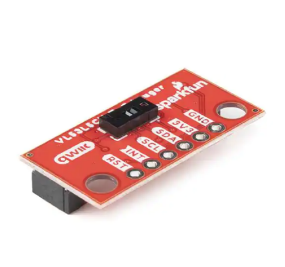


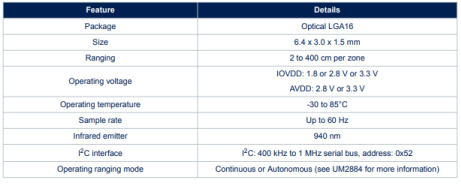
### Proximity Sensor

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

### 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.

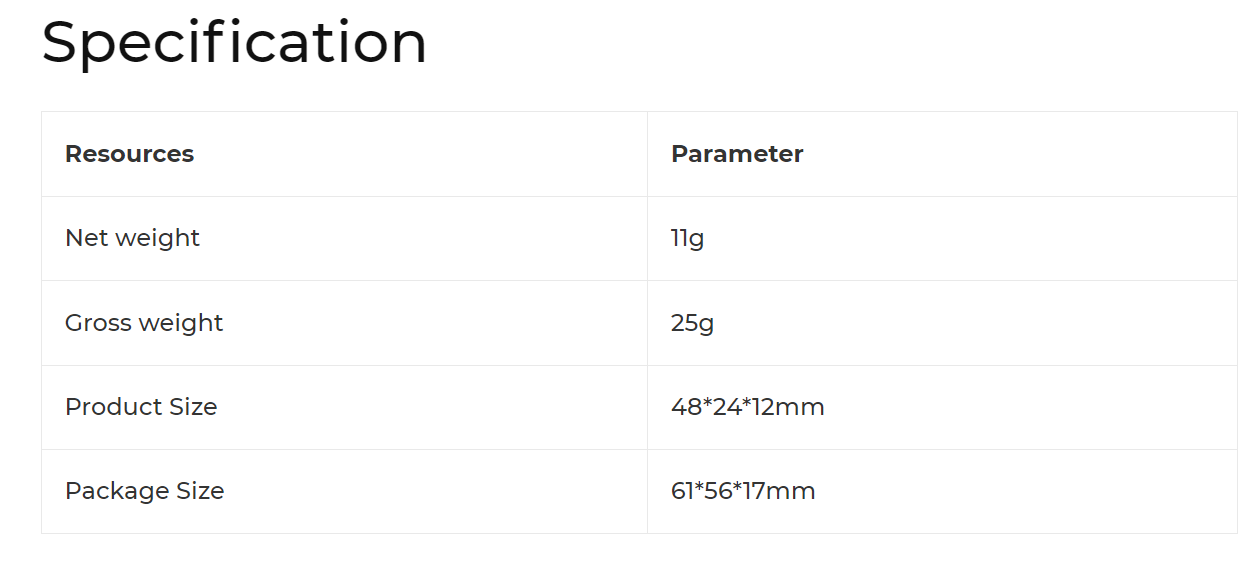
### LCD

Model ILI9342C

### Vibration Motor

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

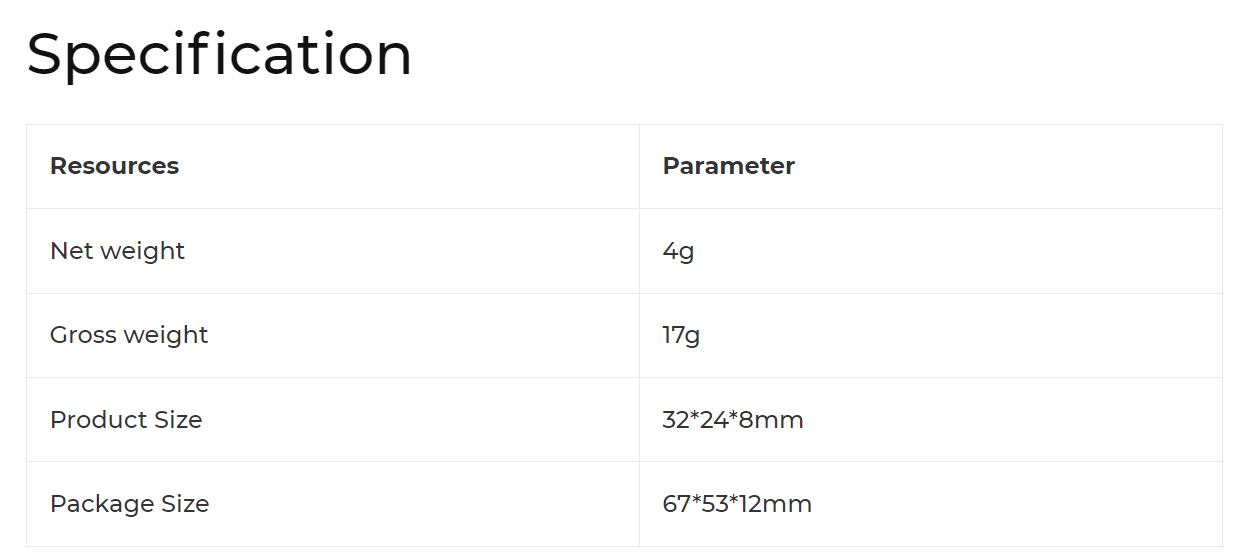
This N20 motor is has a 5V supply voltage. The output shaft has a rotational speed of 8800 RPM. Specifications can be seen below.



### 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.



## Software

### Installation

#### Arduino IDE

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

Libraries required:

1. M5CoreS3
2. Adafruit AMG88xx
3. Adafruit NeoPixel
4. SparkFun VL53L5CX Arduino
5. MAX30100

#### Python 3.11

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

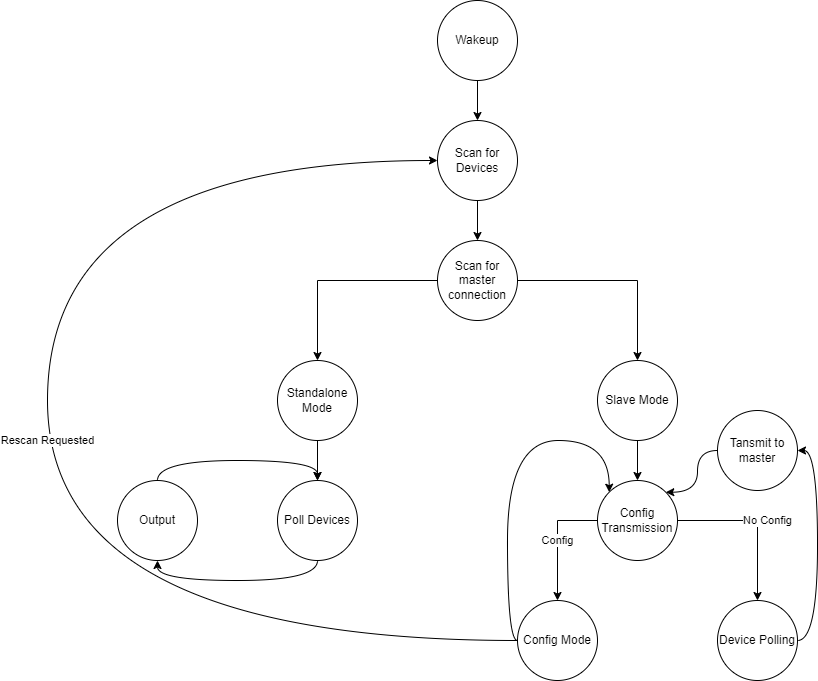
#### PyCharm

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

Libraries required:

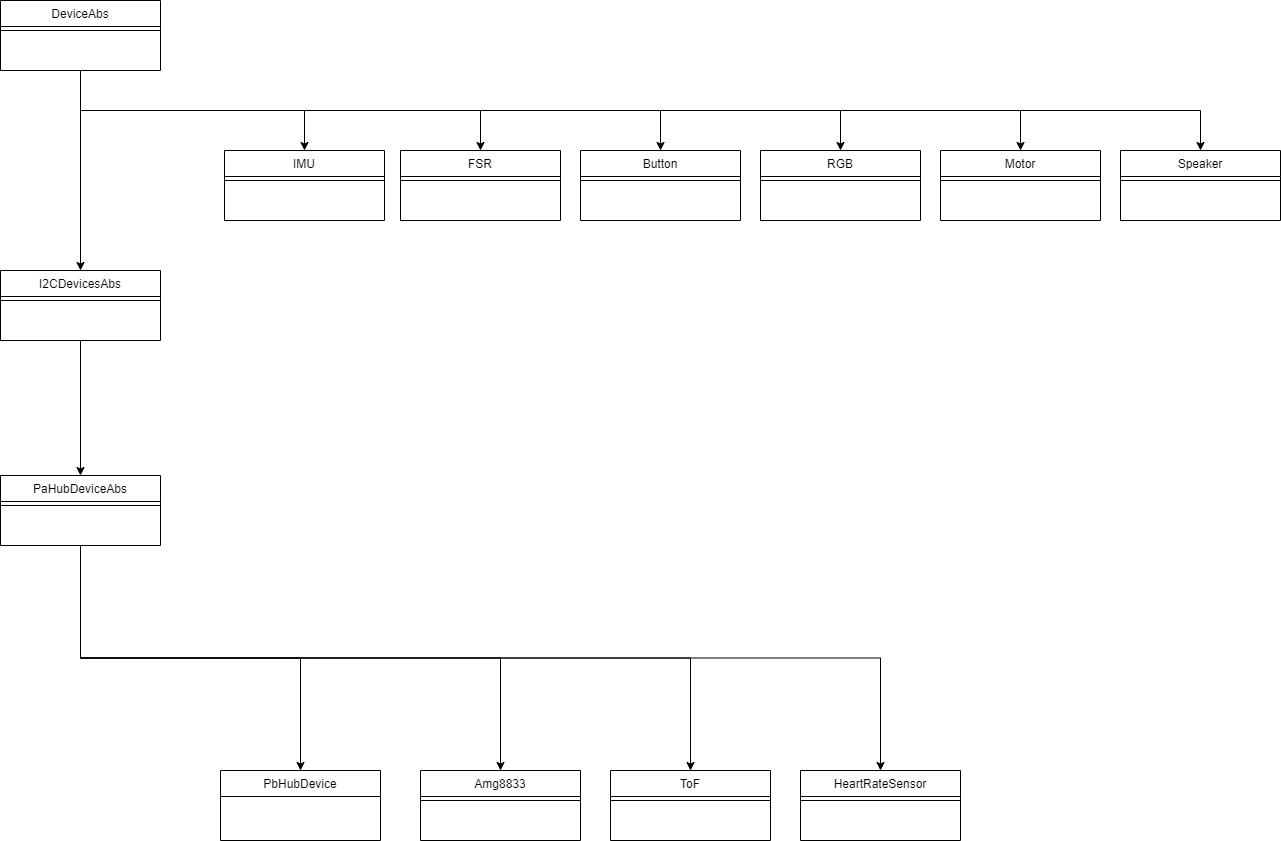
1. Pyserial
2. Numpy
3. Matplotlib
4. Pywin32
5. Scipy

### State machine



### Design

#### Device



### M5Telemetry

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

#### Command Handler

An API which handles connect via Serial / WIFI in case of slave mode and sending buffer to server ( M5Stack device is the **client**).

### Standalone

Standalone will run in 2 cases:

1. Force standalone run.
2. No master 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!**

### Serial

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

Required Arduino code:

#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

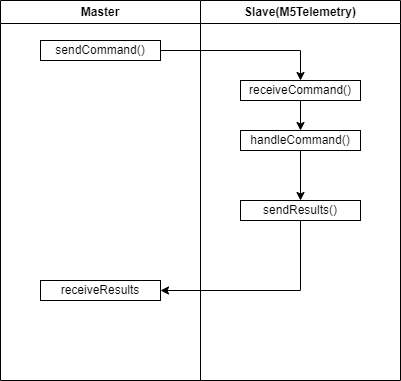
        false                     // use RGB device(Supported only in PORT B)

        );

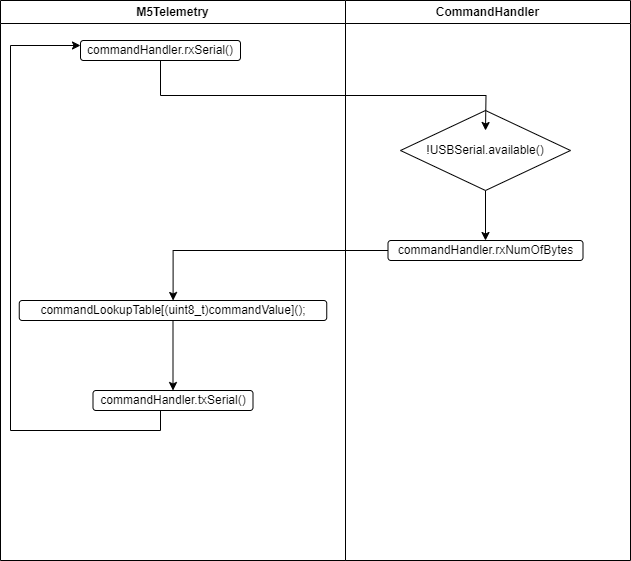
}

#### Flowchart

HDL



Command handling after connection established



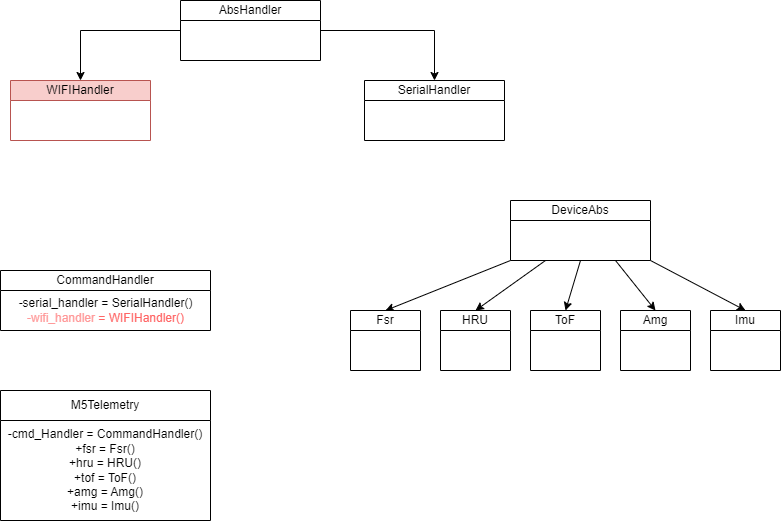
The are two buffers for command handling:

RxBuffer – buffer which holds words (arguments values) which received from master.

TxBuffer – Buffer which holds raw data from device. The parsing of the buffer needed to be done by user (An example is COMMAND\_RUN\_SENSORS).

#### Python(Master)

#### Class

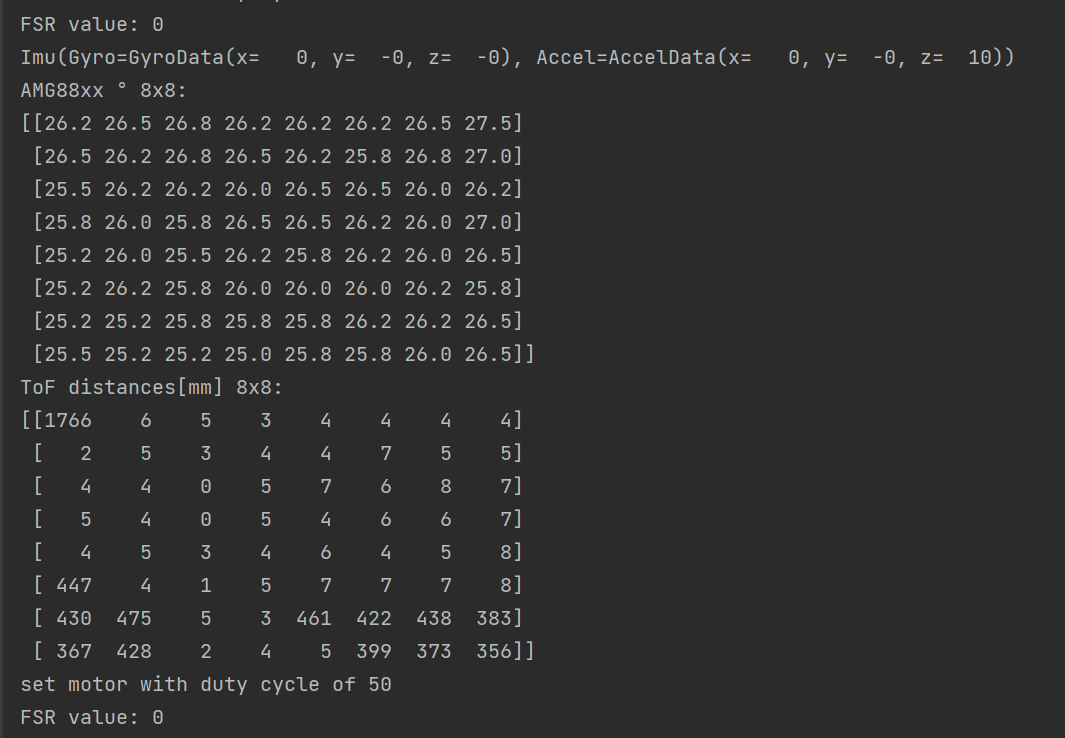


By using PyCharm data from device may be achieved. Each command is created in **master** must be aligned with M5Telemetry **API!**

**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:

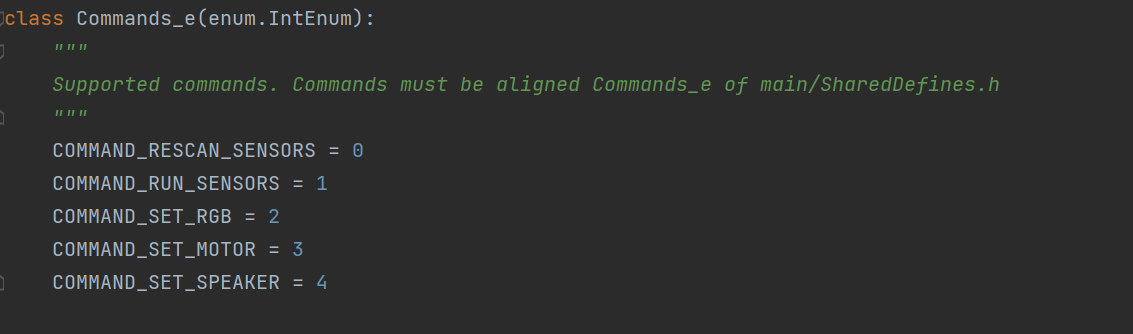


### How to add new command

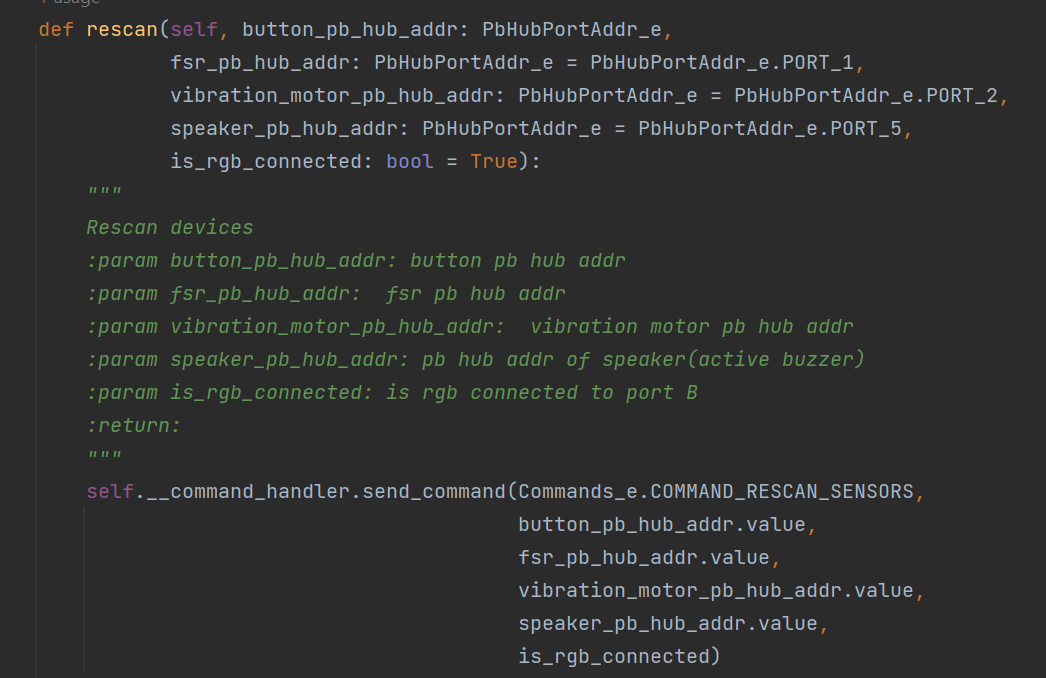
Add command name into 2 Commands\_e enum(master+slave). **Note that enums must be aligned in values otherwise it may lead to unexpected behavior!**

### Master:

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

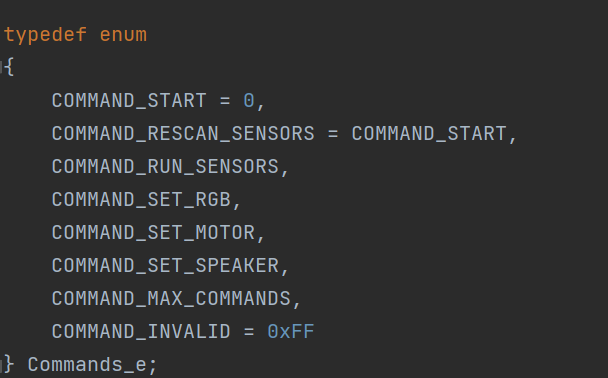


Afterwards create in M5Telemetry the command with specified arguments and pass to send\_command CommandHandler method the command name and the specified arguments:



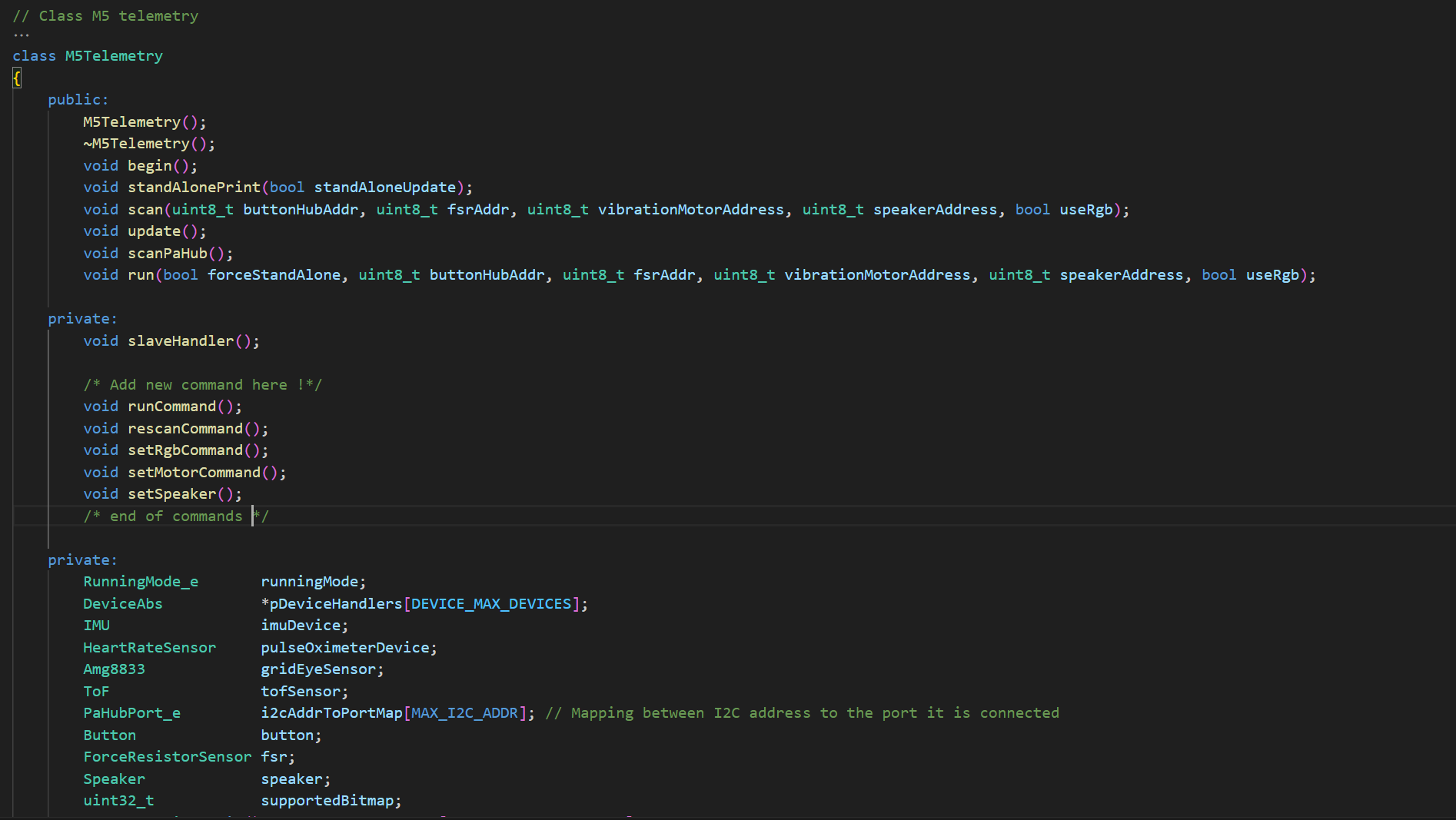
Rescan device example

Slave Commands\_e location: main/SharedDefines.h

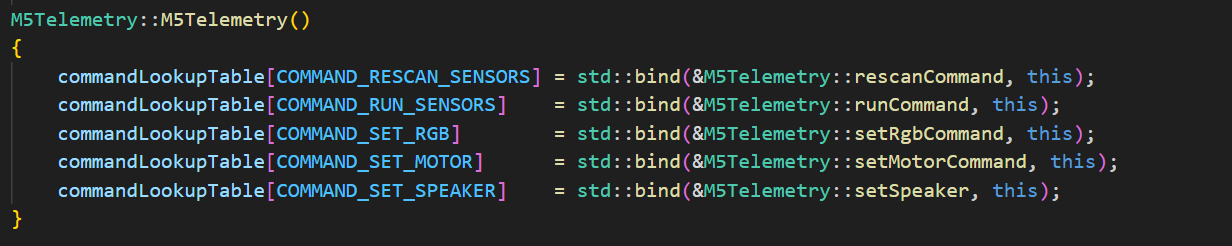


**Note that command must be before MAX\_COMMANDS!**

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 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.

## Bugs

* Due power issues RGB blue only lightup.
* Due power issues vibration motor won't work if to much devices are connected
* Heart rate unit only will work if I2C won't be interfered otherwise it leads to unexpected behavior

## Need to do

* WIFI Handler
* Speaker / Touch via M5Stack(If working API will be provided)