# Final project

M5StackCore3 Telemetry Infrastructure

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 document, we aim to provide a detailed overview of the array of sensors and components available for use in the system. Our primary objective is to establish an ArduinoIDE-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 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 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 factors. First and foremost, the M5Stack belongs to the ESP32 family of microcontrollers. This pedigree not only ensures robust performance with low-power modes but also provides compatibility with a wide range of software and hardware libraries, making it an ideal platform for our project. This comes in shape of Arduino-IDE compatibility and many online libraries useful in Arduino-IDE. The ESP32 also boasts built-in WiFi capability, which can be disabled in order to conserve power using internal registers.

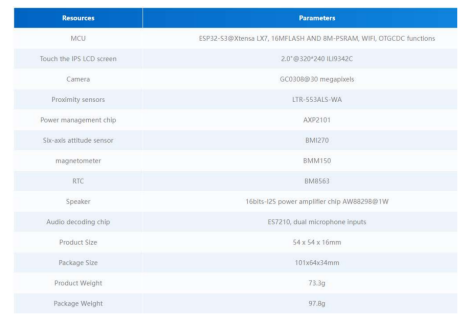
Another noteworthy advantage of the M5Stack is its integrated sensor suite. Several sensors required for our project are already built into the M5Stack, greatly simplifying the wiring and setup process for students. This integrated approach not only enhances convenience but also reduces the potential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

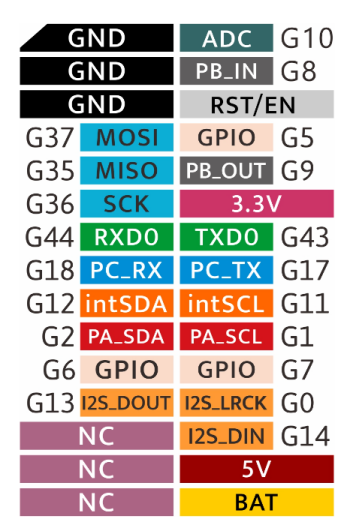
One of the standout features of the M5Stack is its flexibility in accommodating a variable number of sensors. Unlike more rigid systems that necessitate a fixed quantity of sensors, the M5Stack allows us to easily incorporate and adapt to a diverse array of sensors. This adaptability empowers students 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, and built-in WiFi capability. 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.

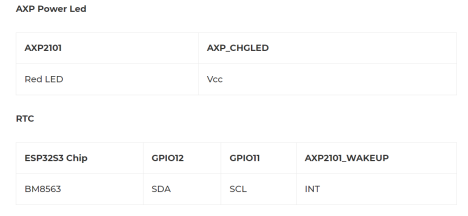
###### System Information



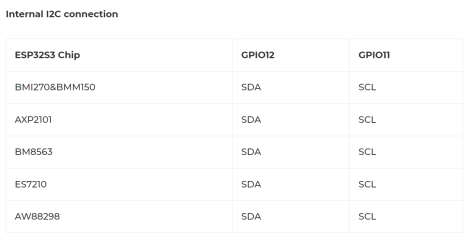
###### Pinout











* 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

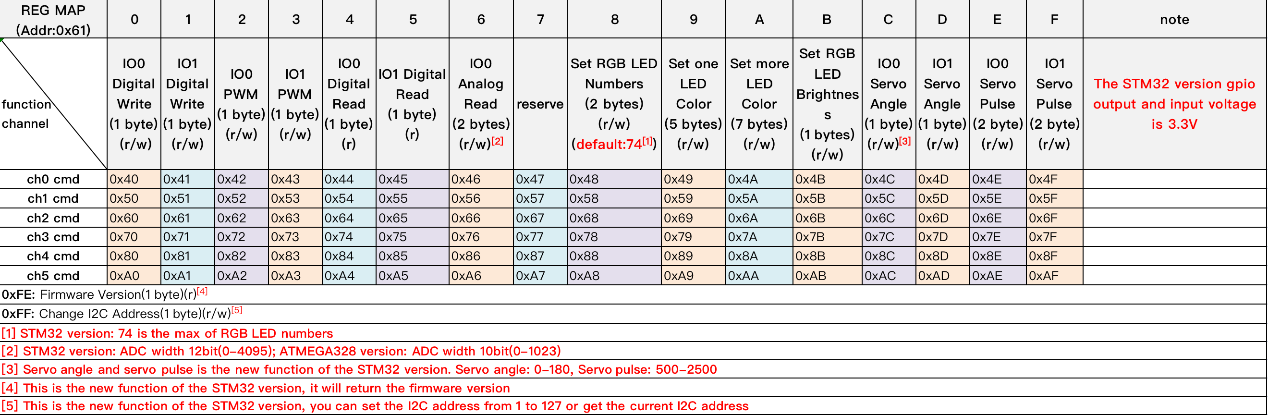
PbHUB Unit is a 6-channel expansion Unit with I2C control. Each Port B interface is capable of GPIO, PWM, Servo control, ADC sampling, RGB light control and other functions. Adopts STM32F030 for internal control.

PbHUB will be connected to PaHUB.

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

###### Register map

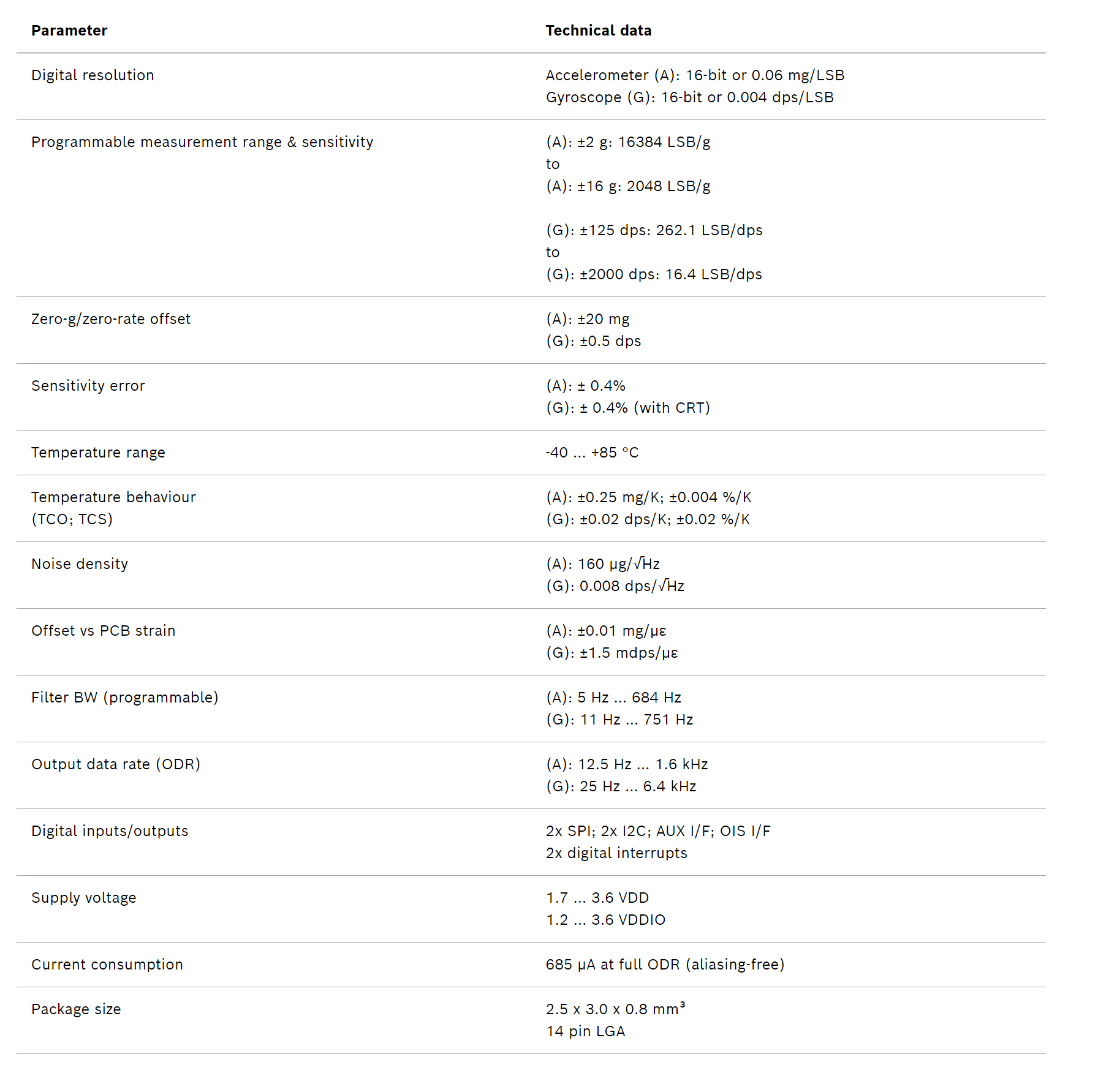
Register map for each port. To set/get register need to use i2c port.



##### 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.

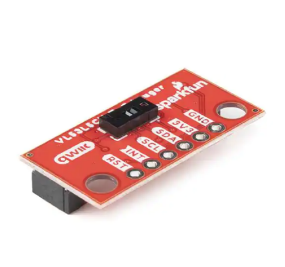


##### Proximity Sensor

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 the handy Qwiic system, no soldering is required to connect it to the rest of your system. However, is still has 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 the handy Qwiic system. However, it still has 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

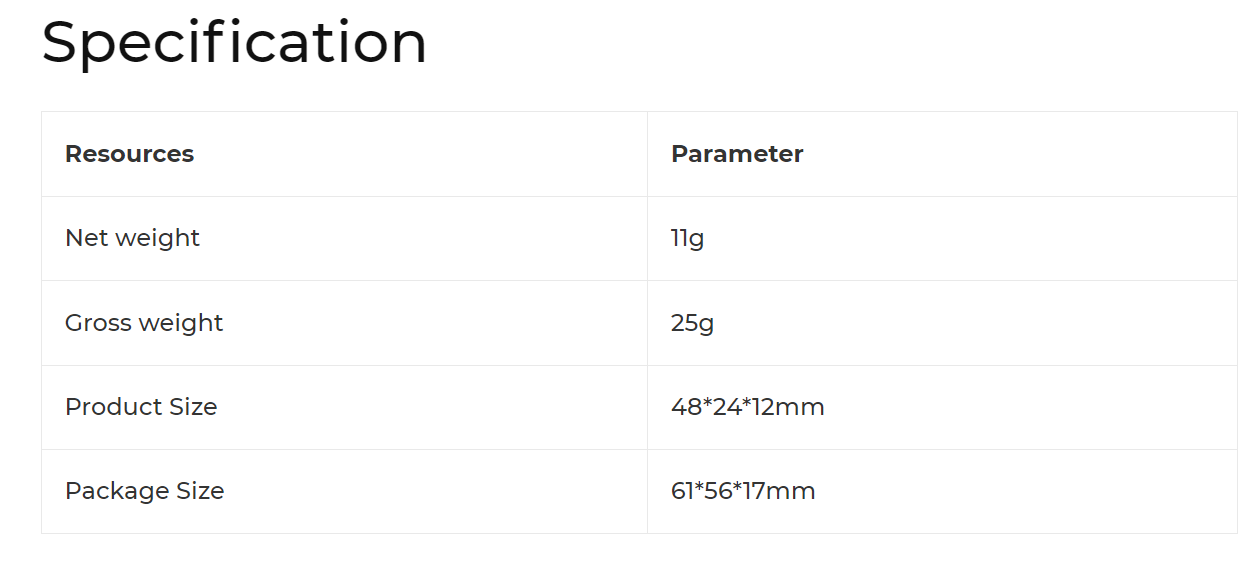
Model ILI9342C

Built Into the M5stack is a Model ILI9342C LCD Display, useful for printing debug and user-facing information mounted on the robust chassis of the Core3.

##### 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.



## Equipment Testing

### Planning

**IMU (Inertial Measurement Unit)** -

The inertial measurement unit is located inside the M5Stack main body, and is in charge of measuring acceleration and tilt in a 6 axis configuration. It’s useful for measuring movement such as the device moving or tipping over, shaking, etc.

The tests run on this device include static tests and dynamic tests. In static tests, we shall place the device on one side, and expect to receive 1g from a single acceleration axis, while receiving 0g from the other axes. In the dynamic tests. We will change the placement of the device and shake it to detect the dynamic response of the sensor.

**Proximity Sensor** **(TOF)** -

The proximity sensor is a TOF sensor, that works by measuring the time it takes for a beam of light to be reflected to the emitter. The specific sensor used is a 8X8 matrix, resulting in 64 pixels of data per request.

This sensor can be mounted separately from the M5Stack body, and can be useful for measuring the distance of the user from a wall, the speed of incoming objects and their general direction, etc. The data is returned as a 8x8 numpy matrix and can be sampled 15 times per second.

Two tests were run on this device: The walk test and hand wave test.

In the walk test, the tester began close to the device, and walked approximately 2 meters in a room away from the device, back to close proximity and then to the back of the room again. The expected result for this is a waveform describing increasing distance and then decreasing distance.

In the hand wave test, the tester stood outside the sensors scan trajectory and waved his hand in front of the sensor. The expected result is indicative of an object suddenly obscuring the view of the sensor and should resemble a square wave.

**Thermal Imaging unit** **(AMG)** -

The Thermal imaging unit is a sort of camera that returns a temperature signal. Similar to the TOF sensor, it is a 8X8 pixel matrix, returning 64 pixels of information regarding temperature.

We tested this unit by having the tester stand in front of the camera covered by cloth, then drop the cloth to reveal temperature closer to body.

**FSR (Force-Sensing Resistor)** -

We will measure the FST by having the tester press the sensor rapidly while the M5 polls it in 10Hz sample speed.

### Results

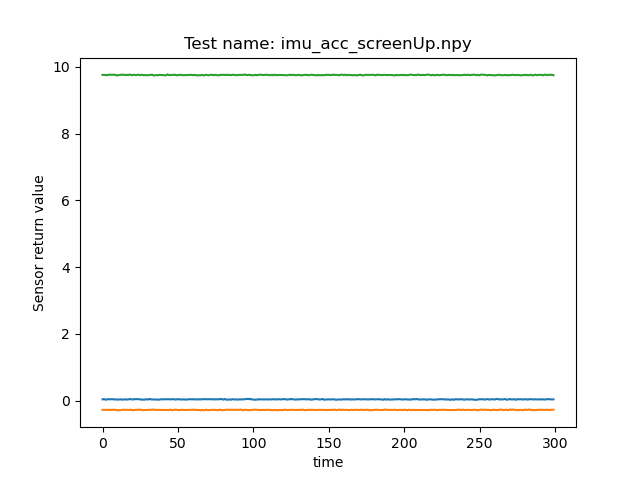
**IMU (Inertial Measurement Unit)**

In the sampling code, we receive the information given by the sensor as an [X,Y,Z] vector. The Z Axis describes the “natural” placement of the device, with it’s screen facing up, while the other axes describe the other faces.

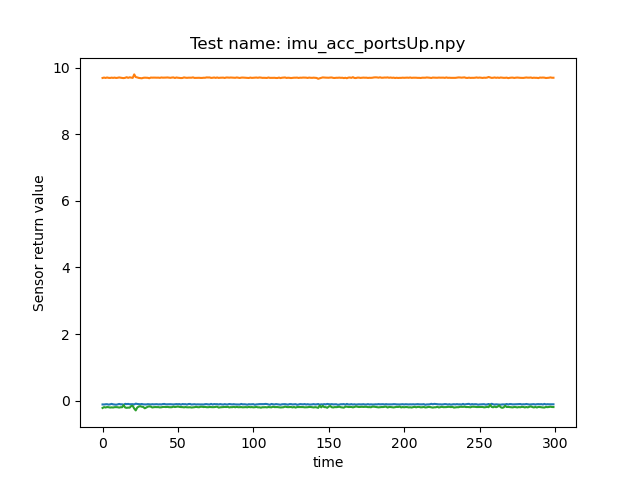
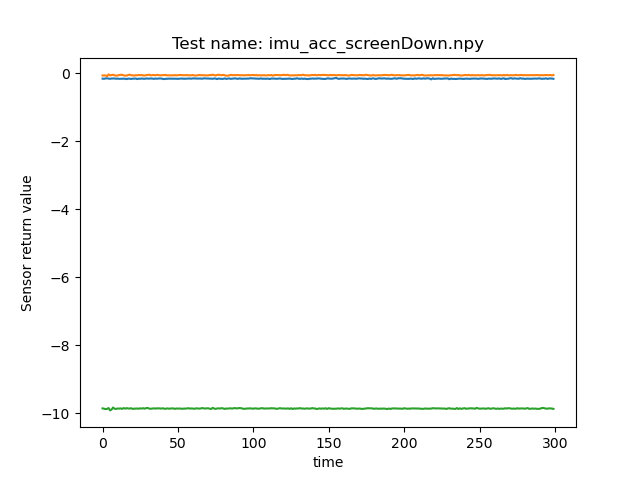
In the following graphs, the [X,Y,Z] vector corresponds to [Blue, Orange, Green].

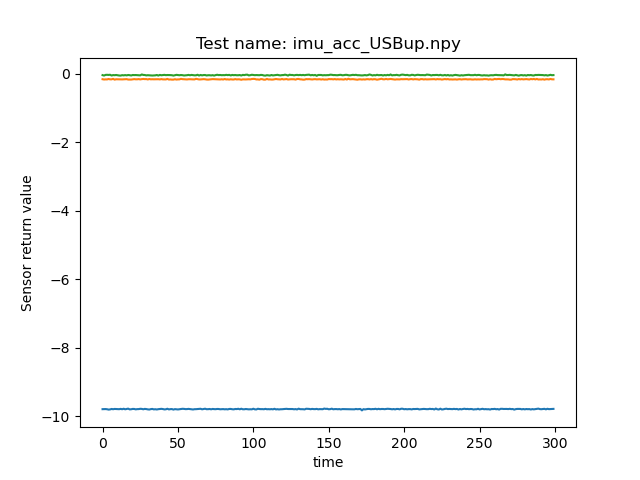
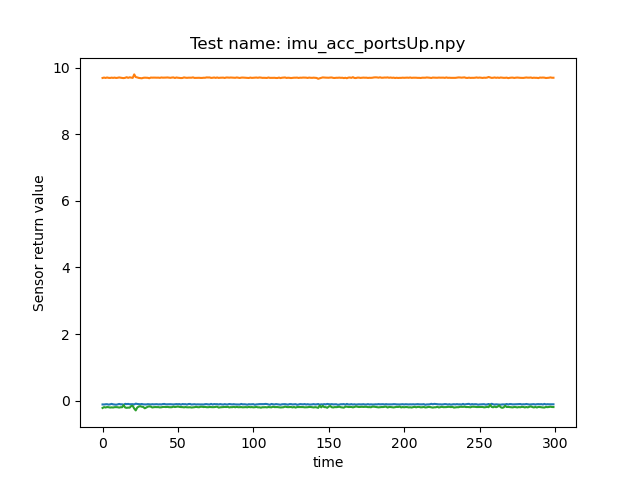
*Static Testing*

Placing the device in it’s natural position, screen facing up, yields the following results.

**

As we can see in this figure, the Z axis shows a result around 9.8m/s^2, while the other axes show results around 0. A similar test was done with the screen down, usb port up, and IIC ports up down.

**

**

These tests can give the user a clue as to what is considered positive and negative in the system.

The screen up position is positive in Z axis.

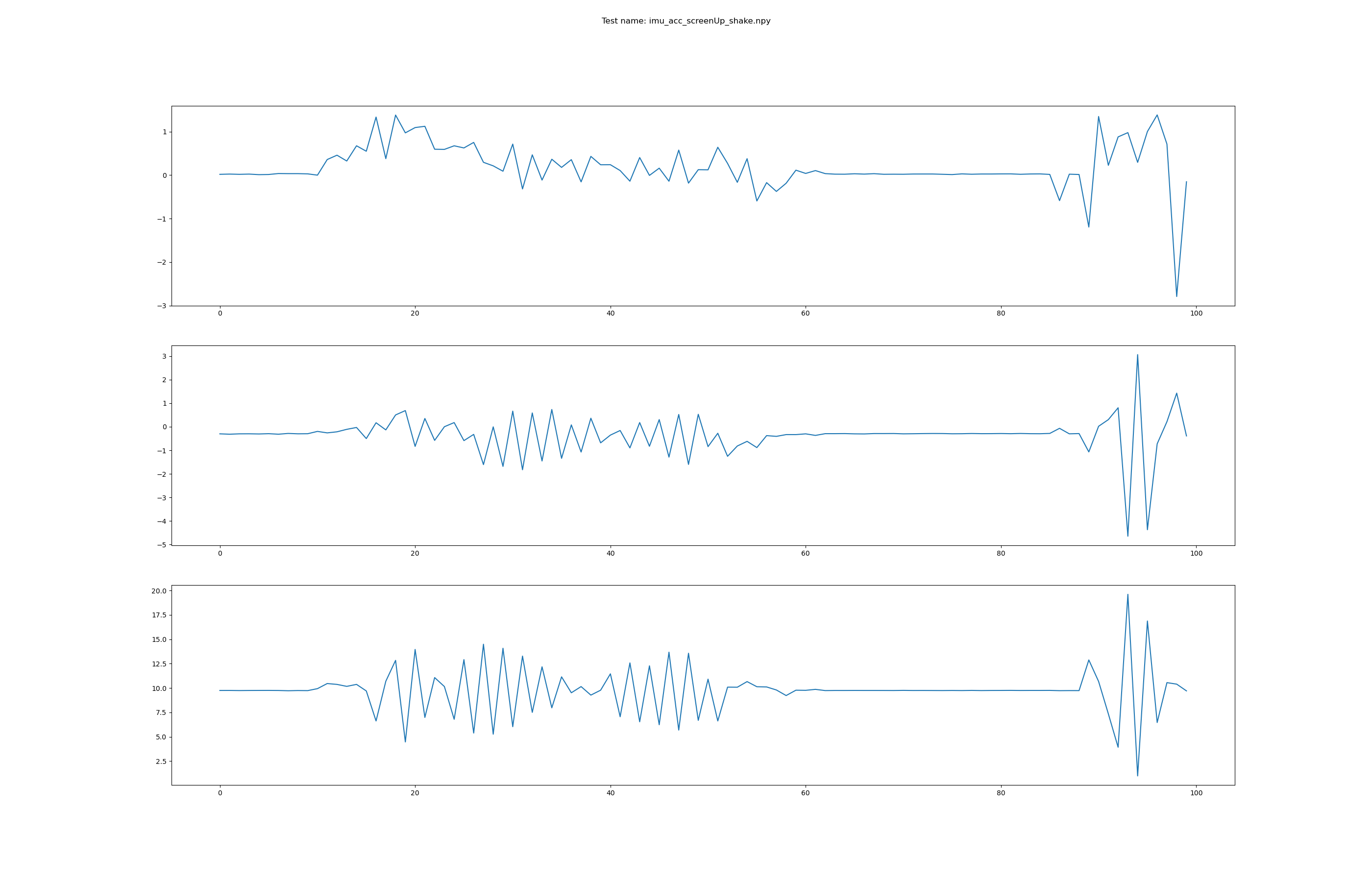
The USB down position is positive in X Axis.

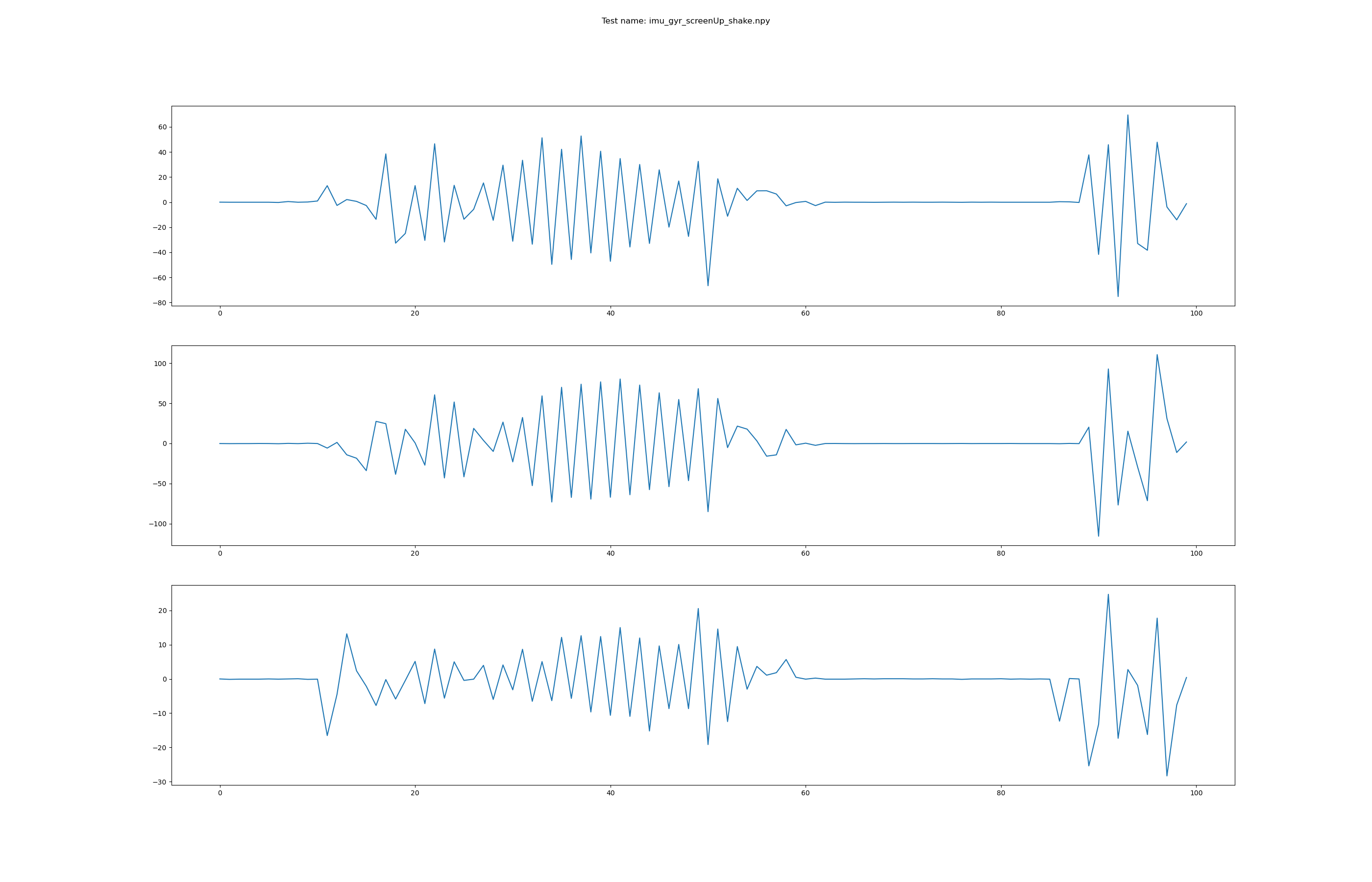
The IIC port up position is positive in Y axis.

*Dynamic Tests*

In the dynamic tests, we did 3 tests.

1. Shake test, screen up.

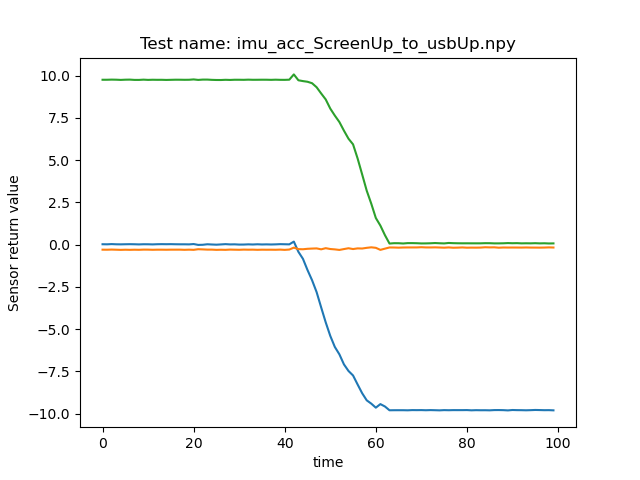
This figure shows results of the accelerometer.

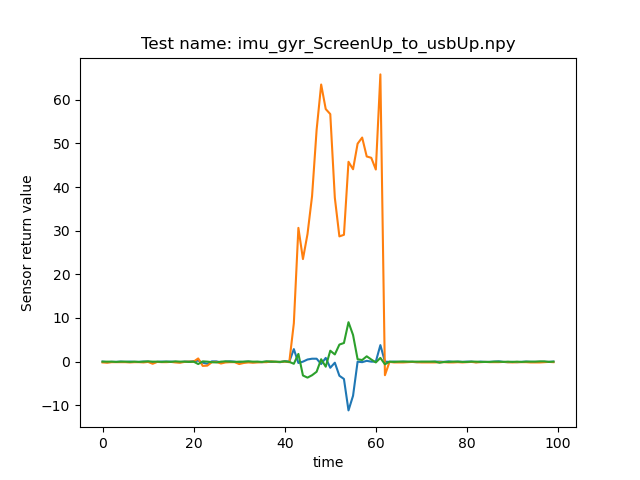
This figure shows the results of the gyroscope.

As we can see, the system can easily detect when the IMU is shaken.

2. Screen up to USB up.

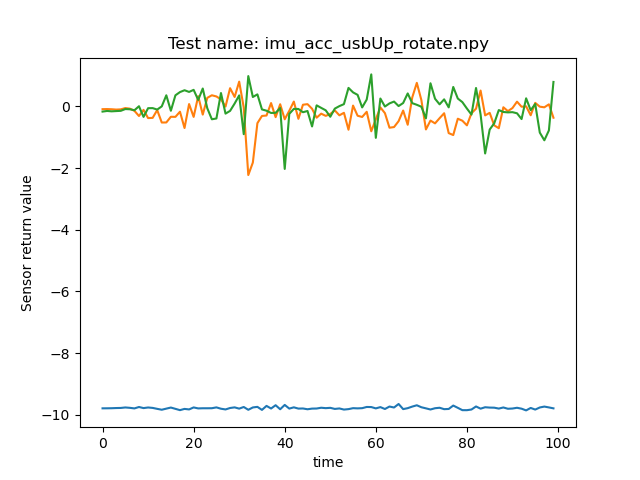
In this test, we lay the M5Stack in a screen up position and tilted to USB-up.

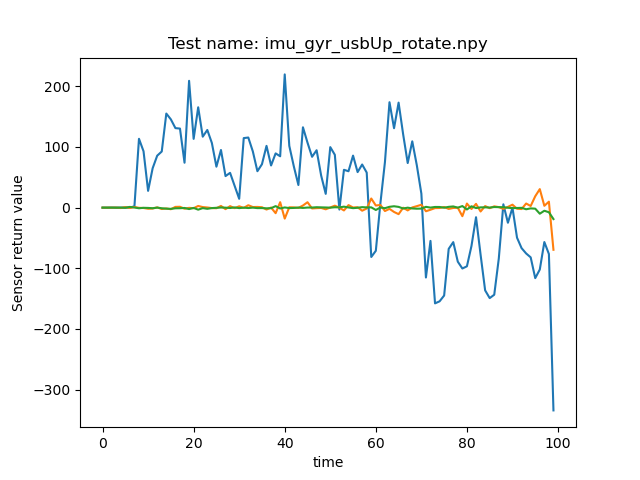


As we can see, both the accelerometer and the gyroscope respond to the movement.

3. USB up rotation

In this test, we held the device with USB-up and rotated it.

**

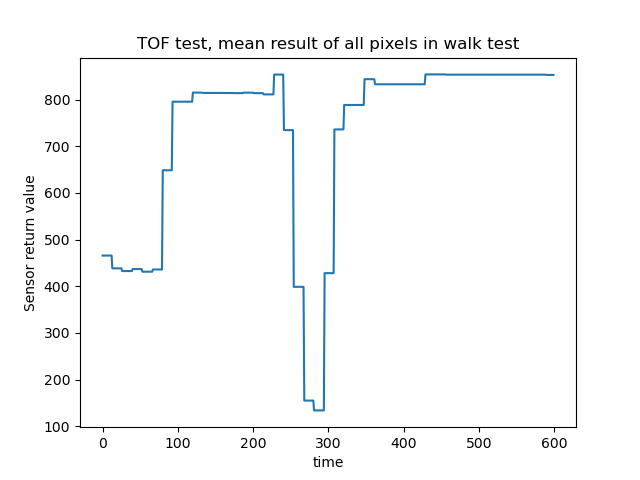
In this case, only the gyroscope shows a response in only one axis, as expected.

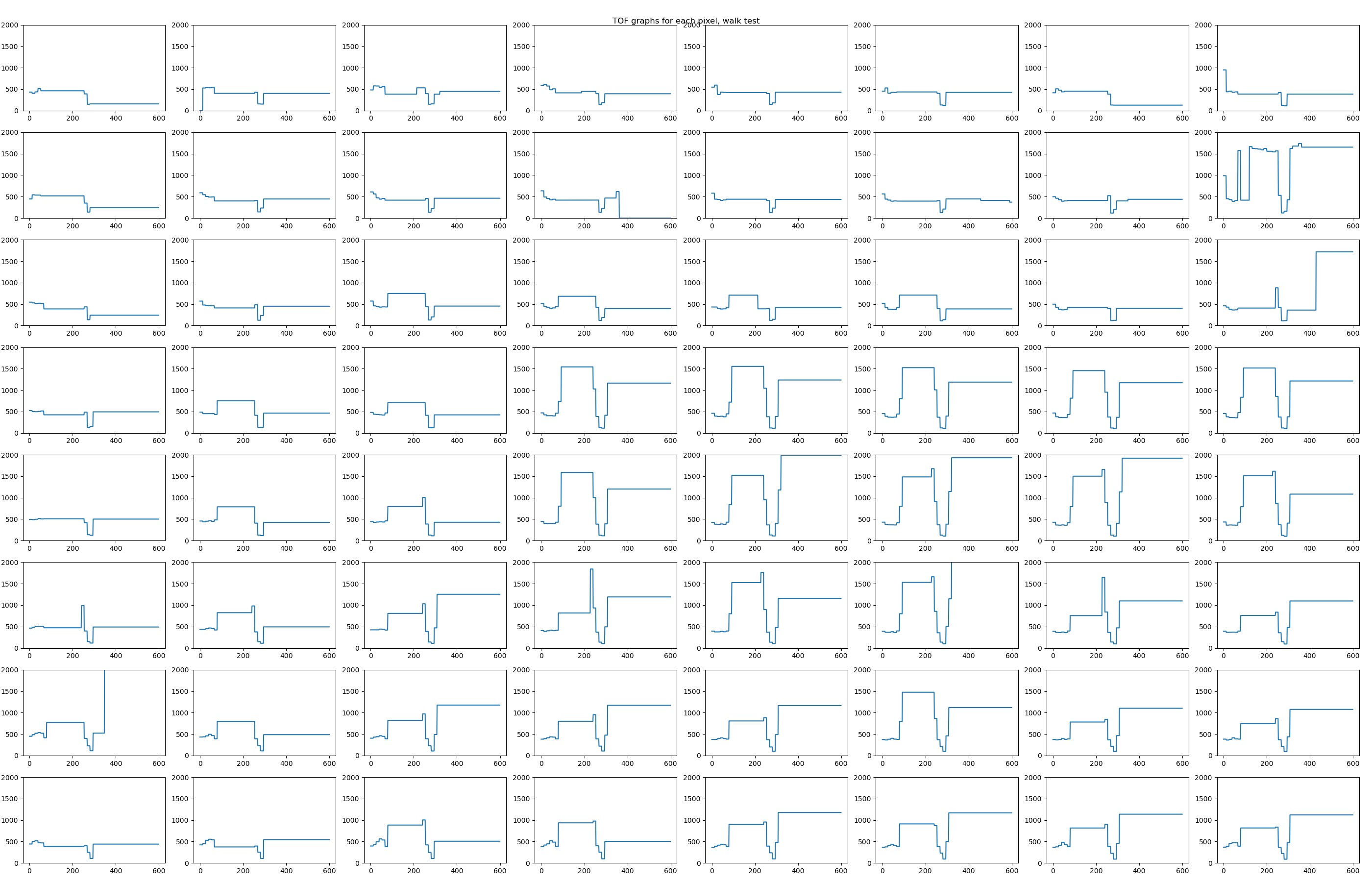
**Proximity Sensor** **(TOF) -**

The tests were done in oversampling to also test the sample rate of the sensor.

*Walk test*

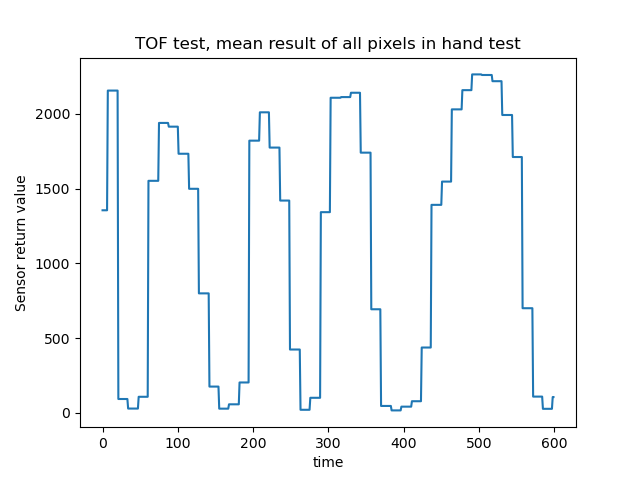
The walk test yields the following results.

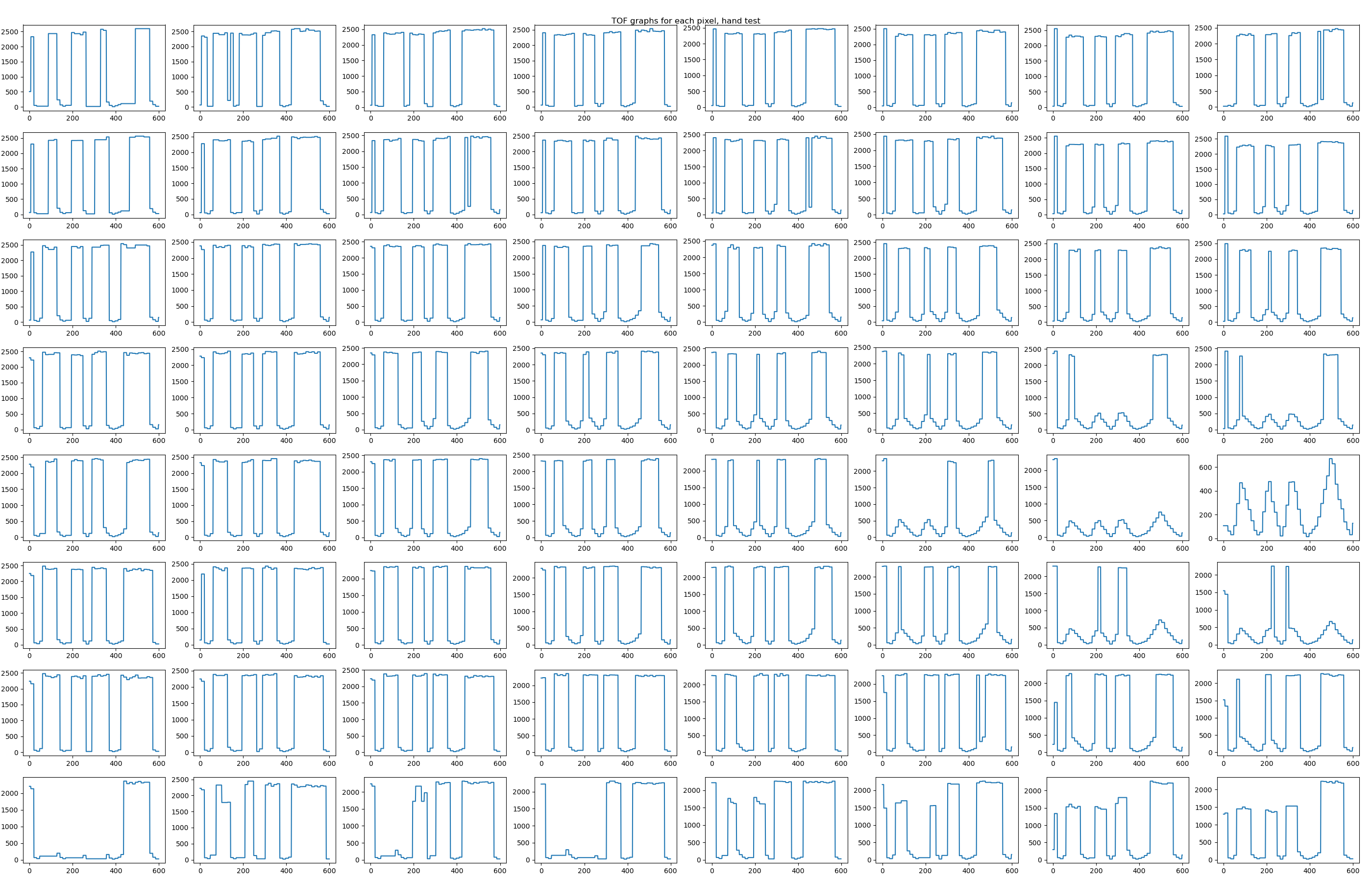
The figure displays the average across all pixels in order to show clearly the increase and decrease in distance. We should note that while a far away body should give high results, a far away body is not seen in all pixels, so the average returned value is not indicative of actual distance. Such use can be useful for detecting close bodies to the sensor. The next figure shows a breakdown of all pixels in this test.

In this display we can see that each pixel responds differently to the tester in this distance. You could discern that the tester is not exactly centered on sensor, but is walking slightly to the right of it as it was in the test due to location constraints. The maximum distance, as can be seen, is larger than 2000mm, as the limits were set to 2000 in the graph.  
This display can be useful for determining the location of the closest object for far away objects.

*Hand Wave test*

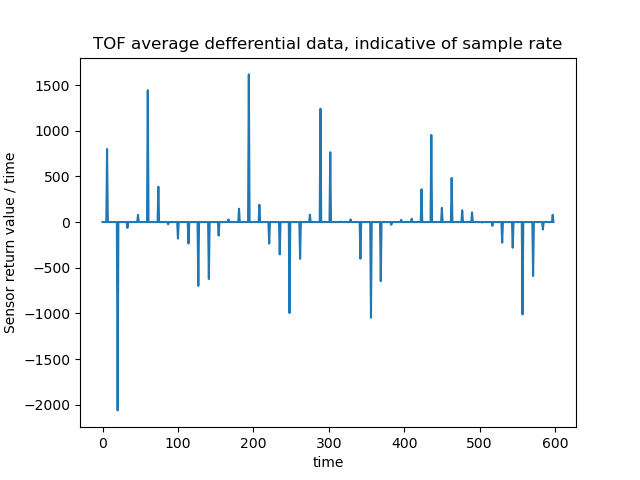
The hand wave results is shown below.

This graph displays the average result of the hand wave test across all 64 pixels. As seen, the low results are returned when the hand is near the sensor, and the high results are returned when the wall is detected instead. A breakdown of the pixel return values is shown below.



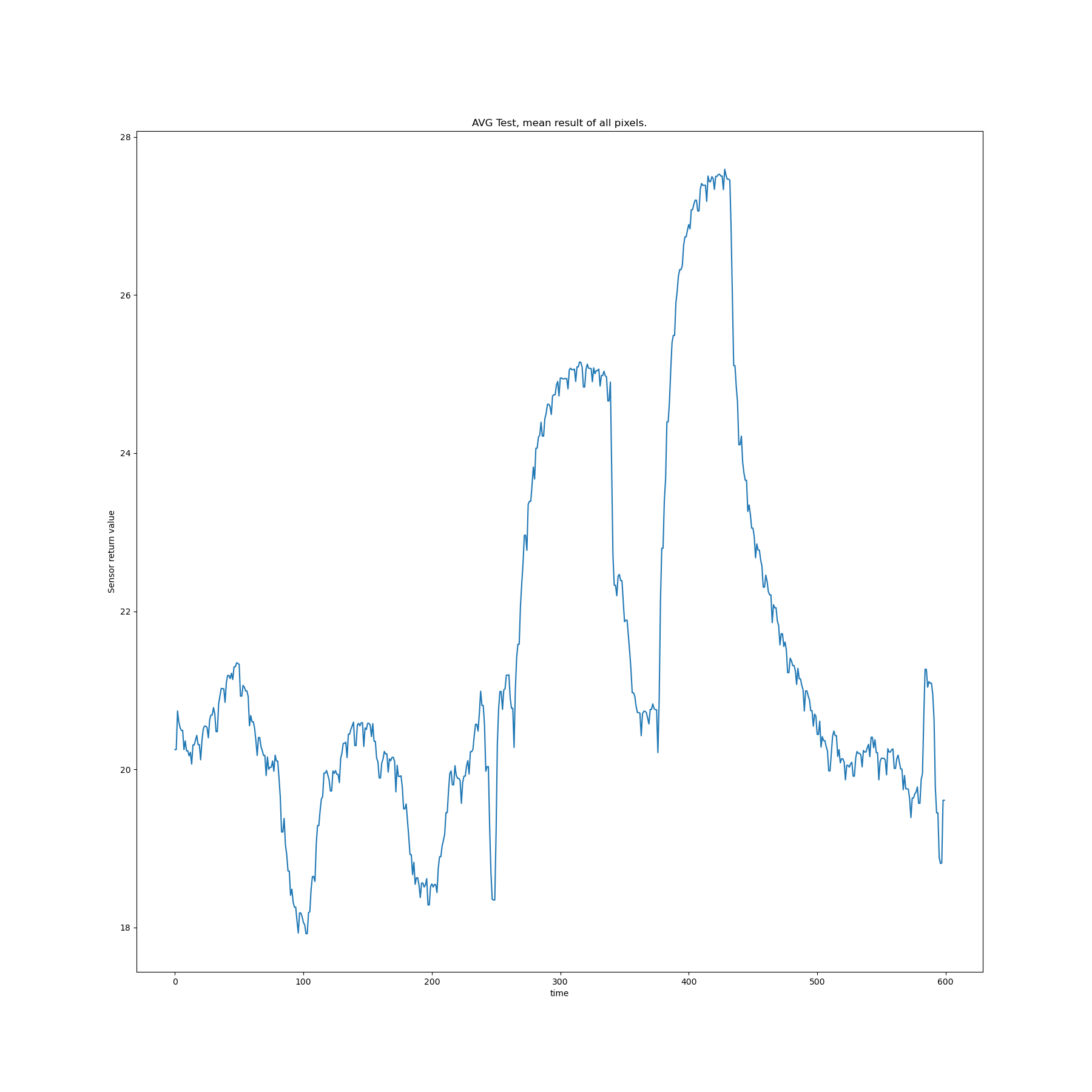
As seen, some of the pixels show less results because of the location of the hand being waved. The upper pixels display less change in proximity because the tester stood left of the sensor and the wrist blocked this pixel most of the test.

We shall now examine the derivative of this graph.

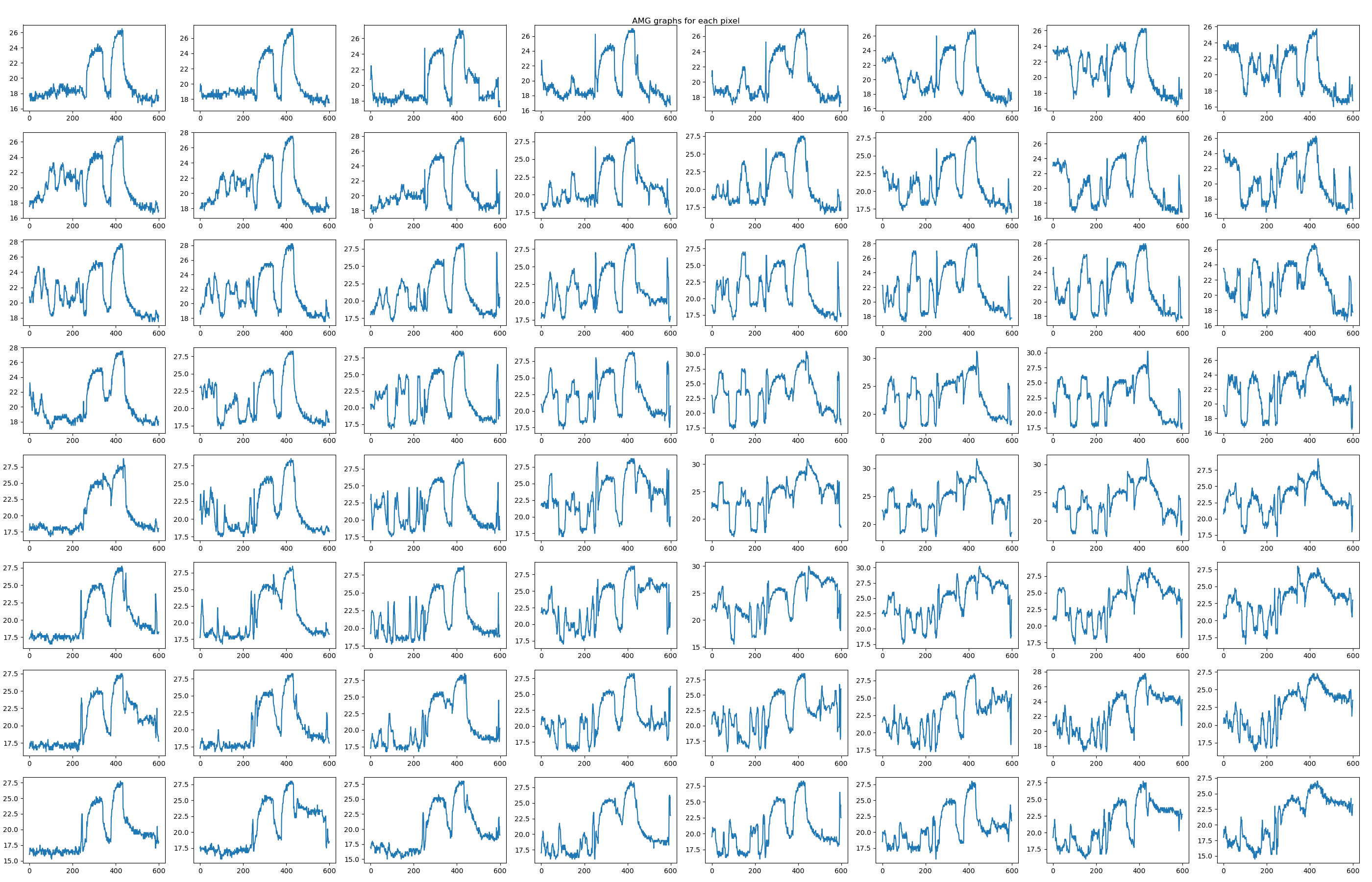
We can see that not every sample updates the graph. In fact, the graph updates 45 times from 600 samples, which is lower than expected.

**Thermal Imaging unit** **(AMG)** -

Similar to the TOF sensor, the AMG test will first be displayed as an average of the pixels.

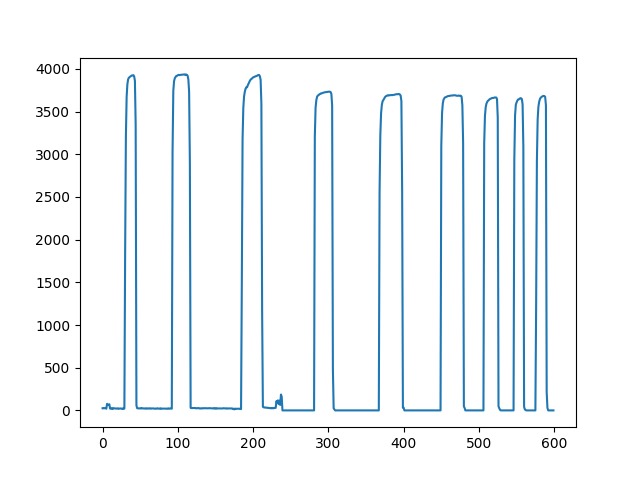
As we can see, the AMG detects an increase in average temperature when the tester lowers the cloth, then decrease when he raises it again.

The pixel images show similar results, without much variation as the tester stood close to the sensor.

This sensor is useful in it’s ability to help with detecting dangerously hot surfaces or items, or in helping determine whether a camera is filming a person or a photo of a person.

The sample rate for this sensor is high, and exceeds the 10Hz sample rate we polled the sensor.

**FSR (Force-Sensing Resistor)** -

Below are the results of the FSR press test.

## 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

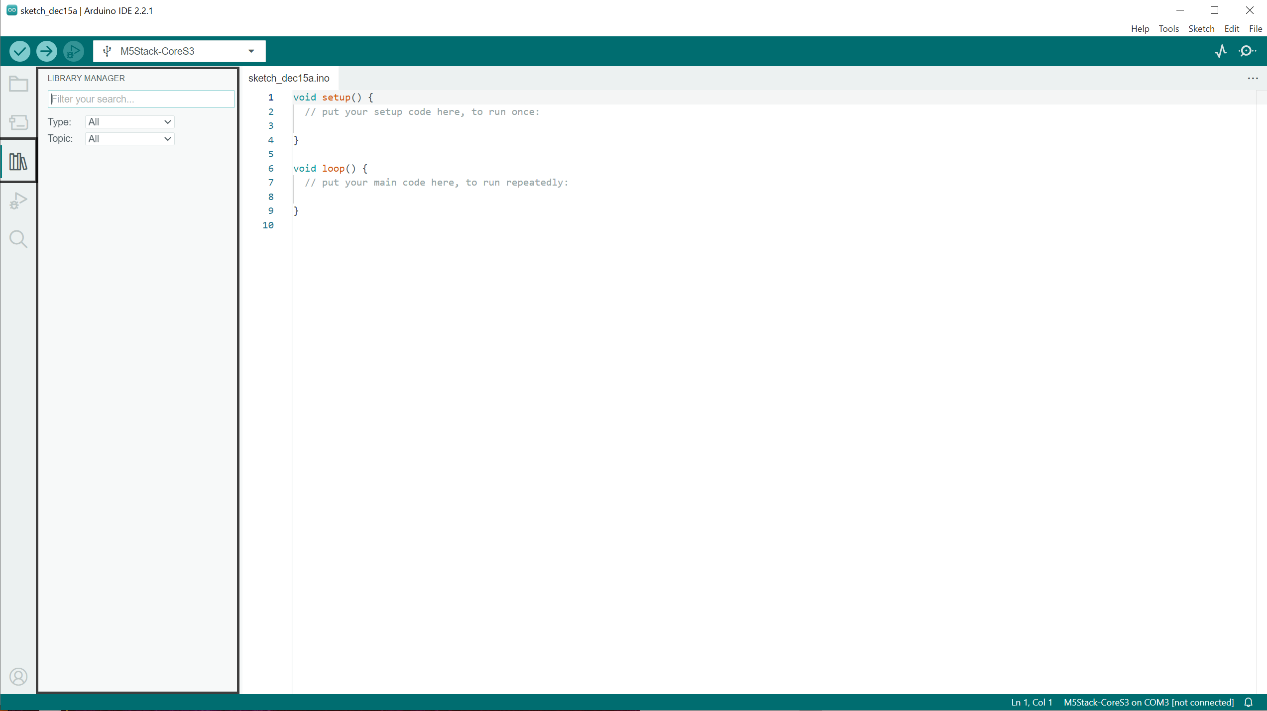
##### Arduino IDE

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

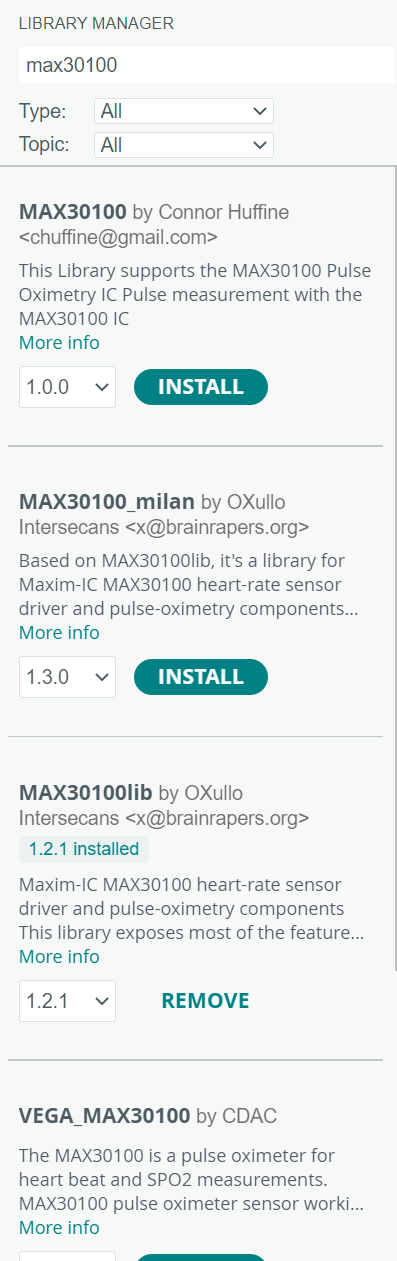
The link below is a tutorial installation provided by M5CORE team regard supporting M5Core in Arduino IDE:

<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.

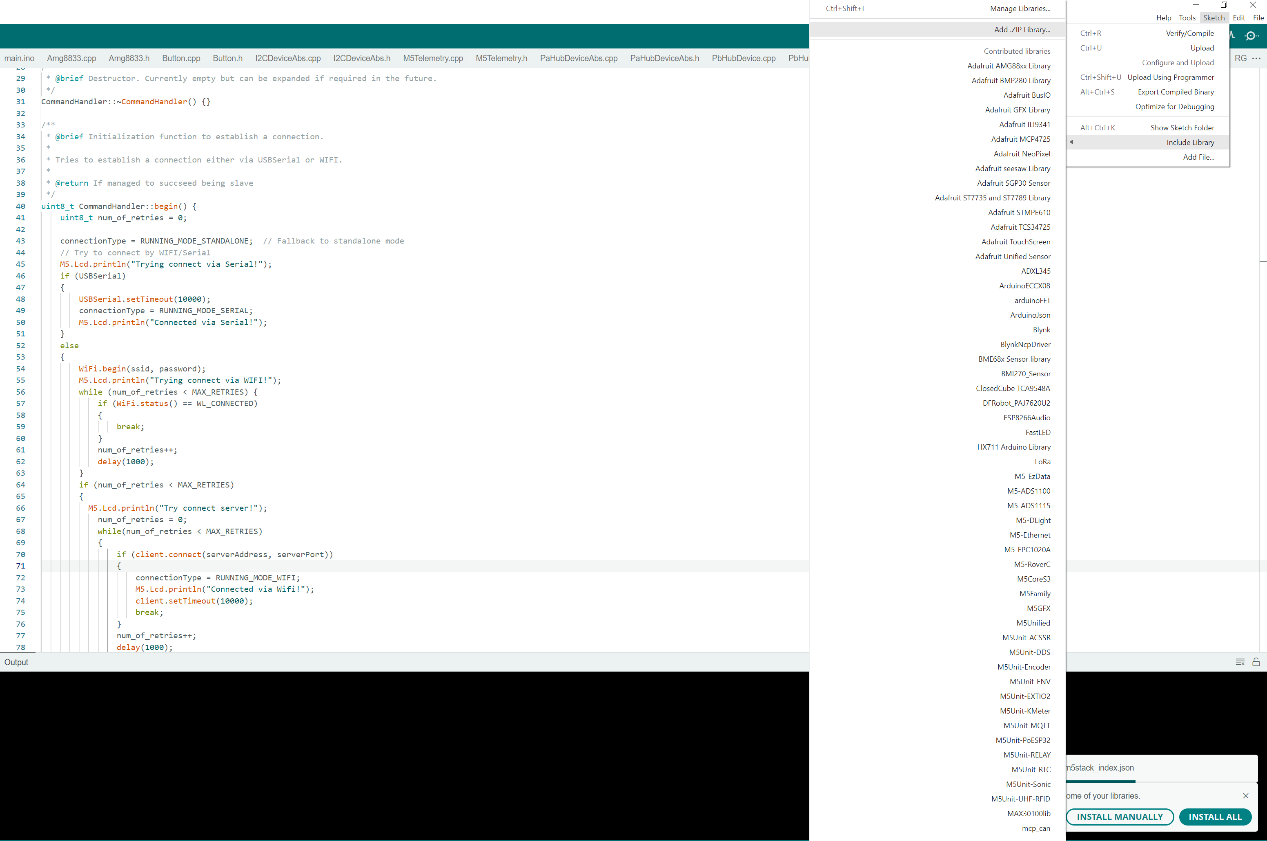
Libraries required:

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

We recommend using PyCharm as a python environment, although any other environment would work just as well, such as Spyder or Notepad++.

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

Libraries required:

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

For easy manipulation of data it’s recommended to use Pandas, but it is not a requirement.

## State machine

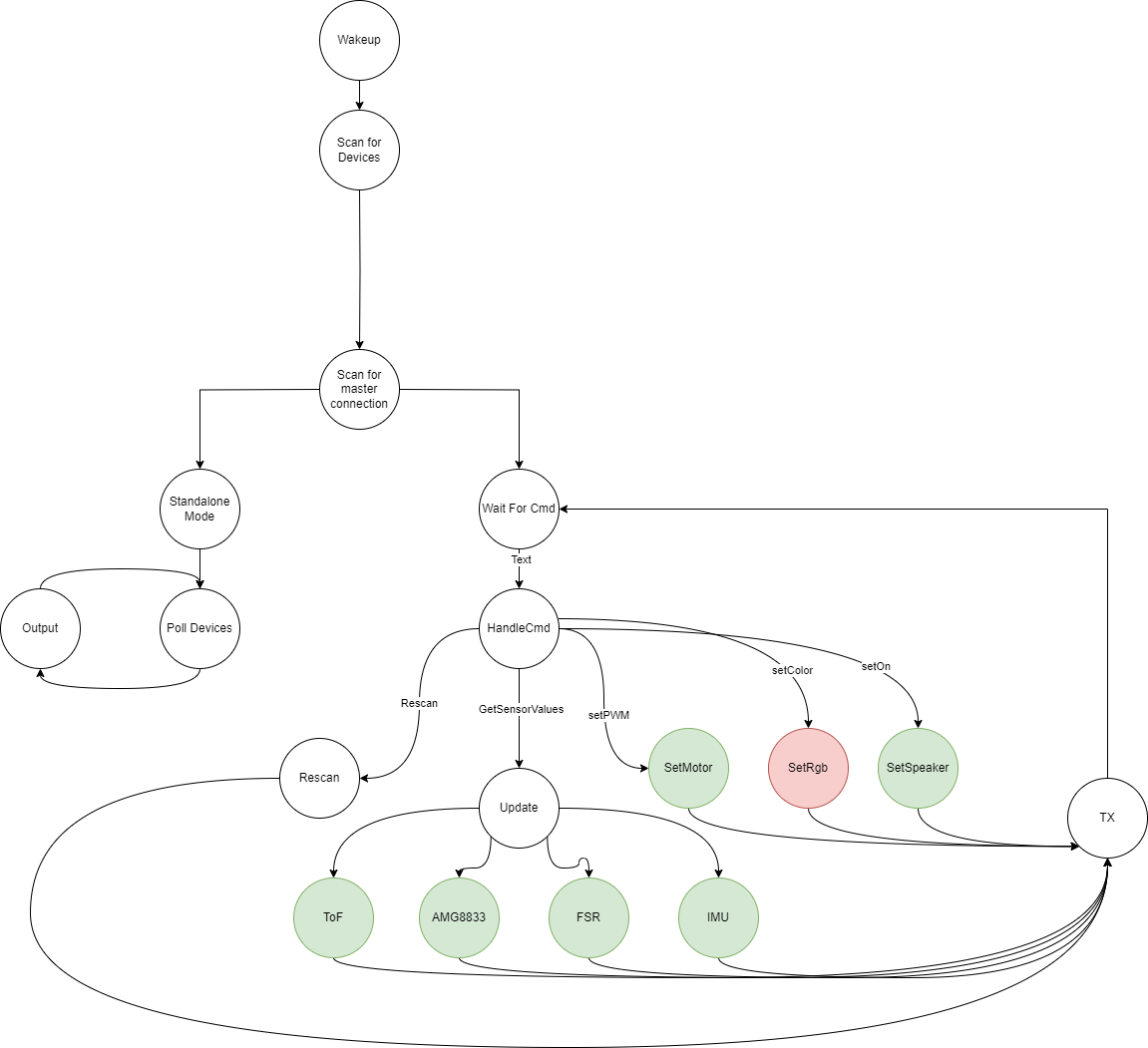
### Arduino

The following state machine shows the state diagram for the system.

In general, the system goes at start-up through a process of scanning the connected sensors. These sensors are connected through PAHub and Pbhub. The supported sensors are the ones given in this documentation, but the system is extendible. After recognizing the sensors, the system will try to boot in slave mode, either through Serial (USB) or WiFi. If the connection is not established after a timeout period, it goes into a rudimentry standalone mode that displays sensor information on the screen.

In slave mode, the user can poll the system using serial commands desctribed in the API section in order to obtain information from discrete sensors, or turn on/off different output devices (such as the active buzzer, RGB or vibrator).

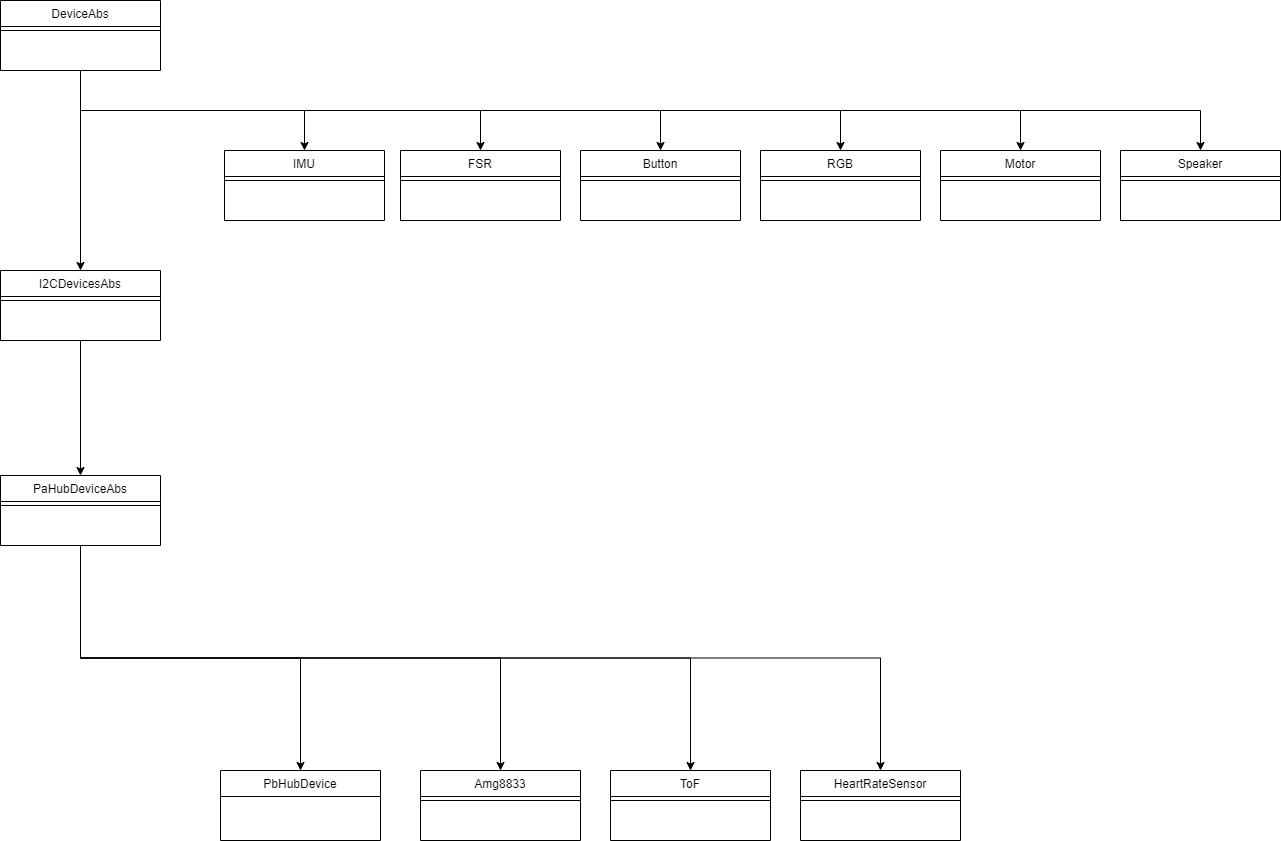
After the information from a sensor is received, the system goes back to standby to TX mode, waiting for more orders.



### Python(Master)

## Software Design

#### Device



## M5Telemetry

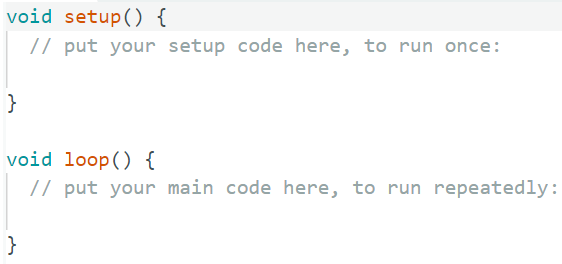
M5Telemetry is the main API which handles standalone / slave mode.

#### Arduino basics

#### Arduino

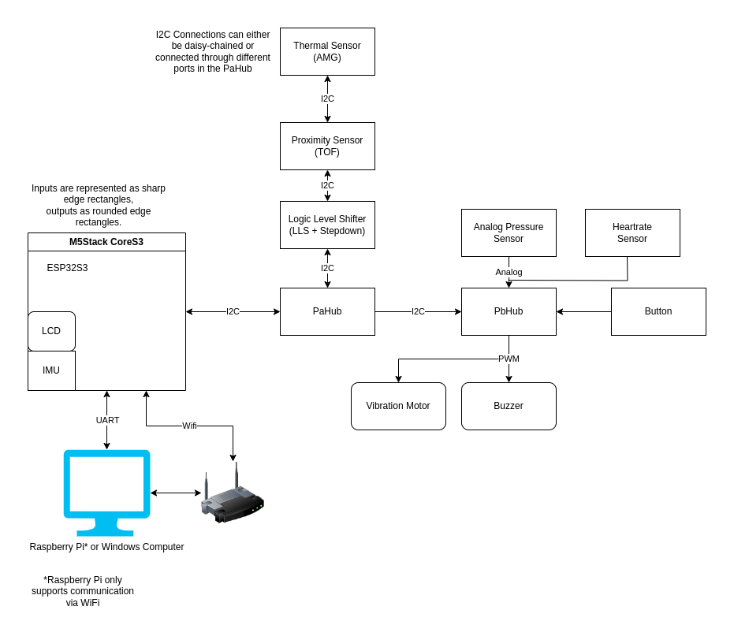
Arduino code is quite straightforward and typically consists of two main phases:

* 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.

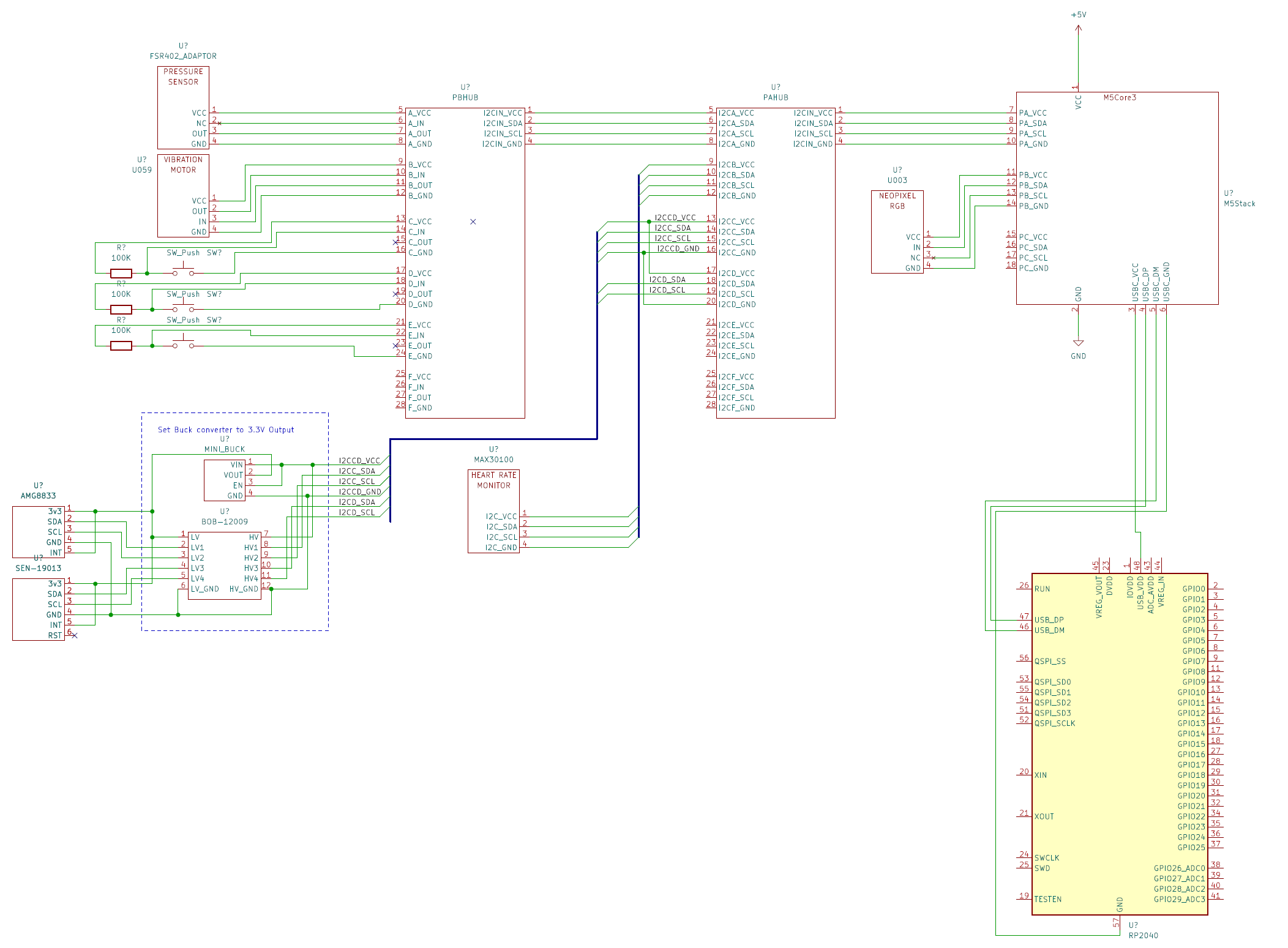


learning the basics of Arduino programming, you can review helpful tutorials on websites like [Tutorialspoint's Arduino section](https://www.tutorialspoint.com/arduino/index.htm).

### 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.

## Usage

#### 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)

        );

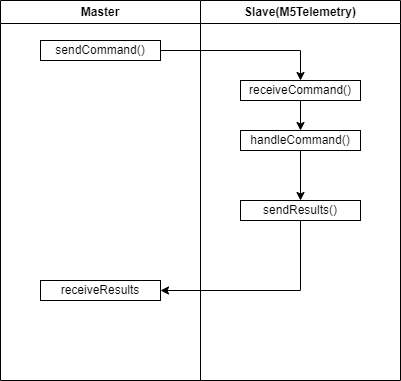
}

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

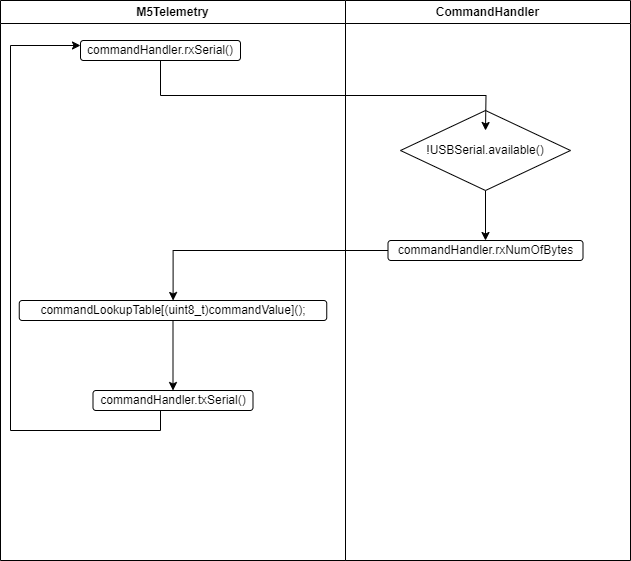


There are two buffers used for command handling:

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 to find a serial connection (via UART) at baud rate of 115200. If successful, a CLI instance will 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

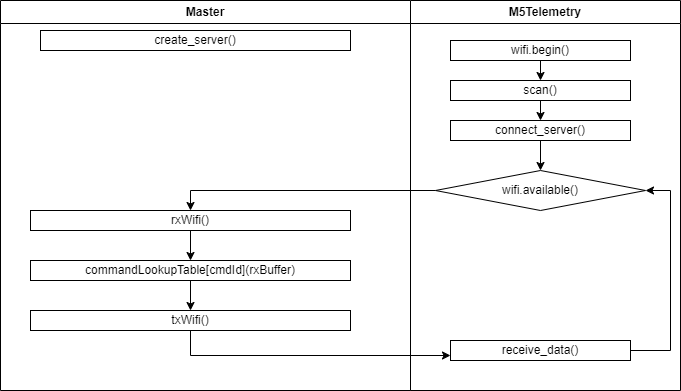
        false                     // use RGB device(Supported only in PORT B)

        );

}

#### WIFI

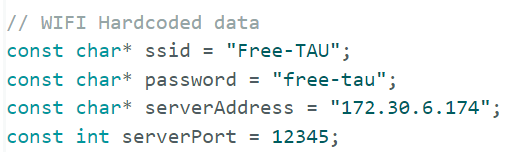
##### Diagram



##### Usage

To connect via WIFI, the user needs to ensure **Serial** connection is closed, or prevent connect directly to PC / Raspberry PI. The Serial connection will always take priority over WiFi.

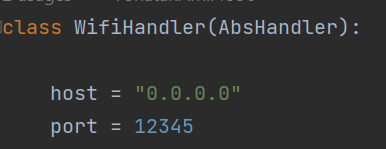
In **<M5StackTelemetry>/main/CommandHandler.c** there's the following section:



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

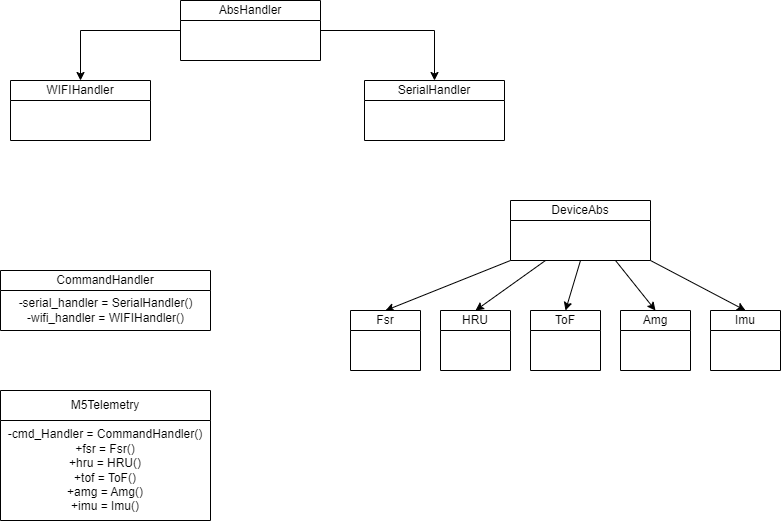


Host needs to be aligned with serverAddress field in M5StackTelemetry FW code.

ServerPort needs to be aligned with port of WifiHandler.

### Python(Master)

#### Class hierarchy



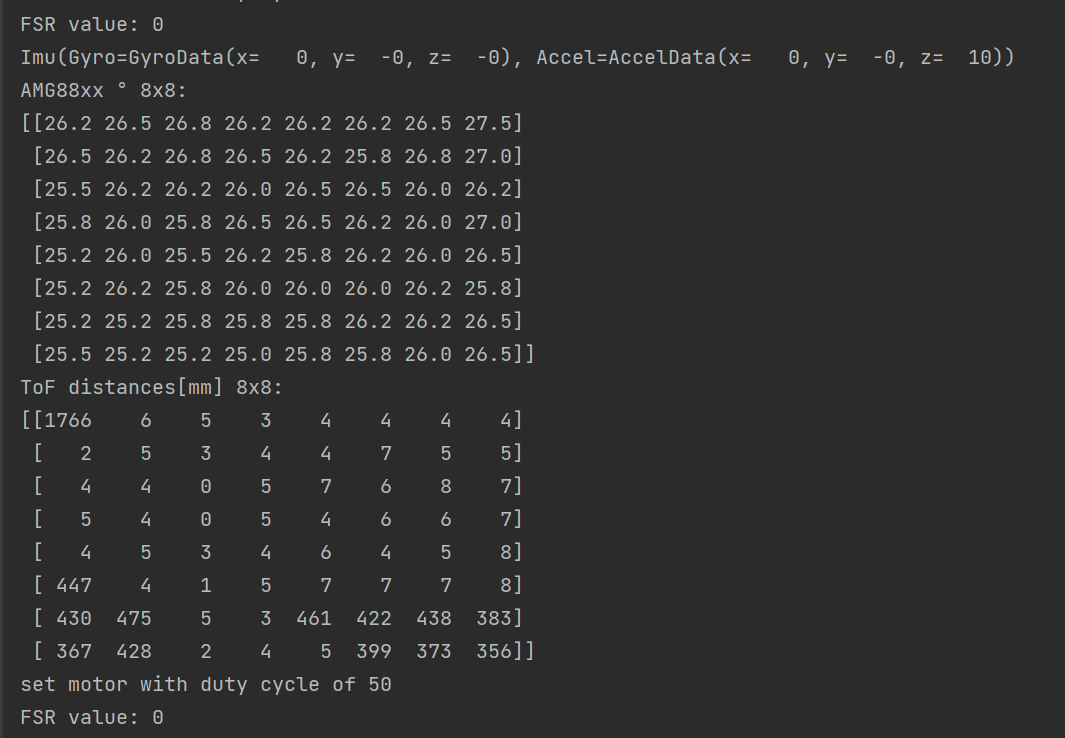
#### Demo

By using PyCharm API - data from device may be polled. 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:



#### Save sensors result in numpy matrix

An example with imu & tof

import os  
import sys  
import time  
import numpy as np  
  
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  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Creating an instance of the M5Telemetry class.  
 interface = M5Telemetry(is\_wifi=True)  
 interface.rescan(fsr1\_pb\_hub\_addr=PbHubPortAddr\_e.PORT\_0)  
  
 imu\_matrix = []  
 imu\_accel\_matrix = []  
 big\_array\_matrix\_tof = []  
  
 for i in range(600):  
 interface.update\_values([Device\_e.IMU, Device\_e.TOF]) # Taking snapshots from TOF and IMU  
 x, y, z = interface.imu.gyro\_data.x, interface.imu.gyro\_data.y, interface.imu.gyro\_data.z  
 imu\_matrix.append((x, y, z)) # Append gyro data  
 x, y, z = interface.imu.accel\_data.x, interface.imu.accel\_data.y, interface.imu.accel\_data.z  
 imu\_accel\_matrix.append((x, y, z)) # Append ACCEL  
 big\_array\_matrix\_tof.append(interface.tof.mm\_distances) # Append ToF pixels  
   
 # Port list to numpy matrix (Sensor output versus time axis)  
 numpy\_gyro\_data\_matrix = np.array(imu\_matrix)  
 numpy\_accel\_data = np.array(imu\_accel\_matrix)  
 numpy\_tof\_matrix = np.array(big\_array\_matrix\_tof)

#### Plot AMG

Plot AMG8833 results:

import os  
import sys  
root\_path = os.path.join(os.path.dirname(os.path.abspath(\_\_file\_\_)), "../../")  
sys.path.append(root\_path)  
  
from CLI.M5Telemetry import M5Telemetry  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 M5Telemetry(plot\_amg=True).plot\_amg(plot\_delay\_in\_seconds=0.01, min\_temprature\_show=10, max\_temprature\_show=40)

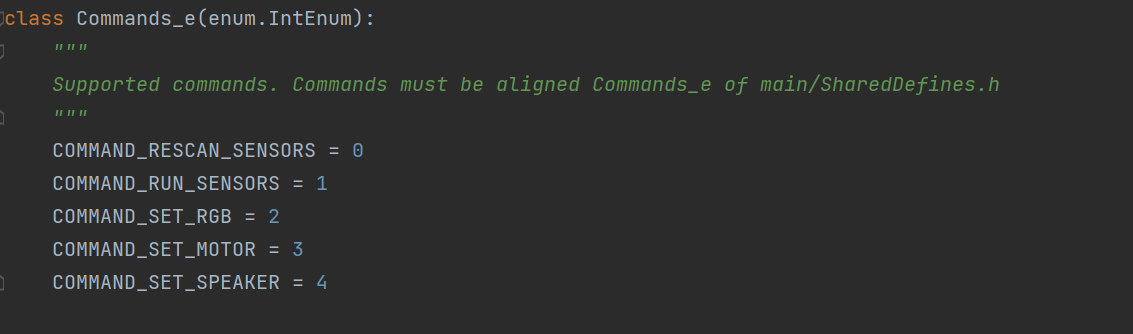
Note arguments may be changed according to your necessity.

### How to add new command

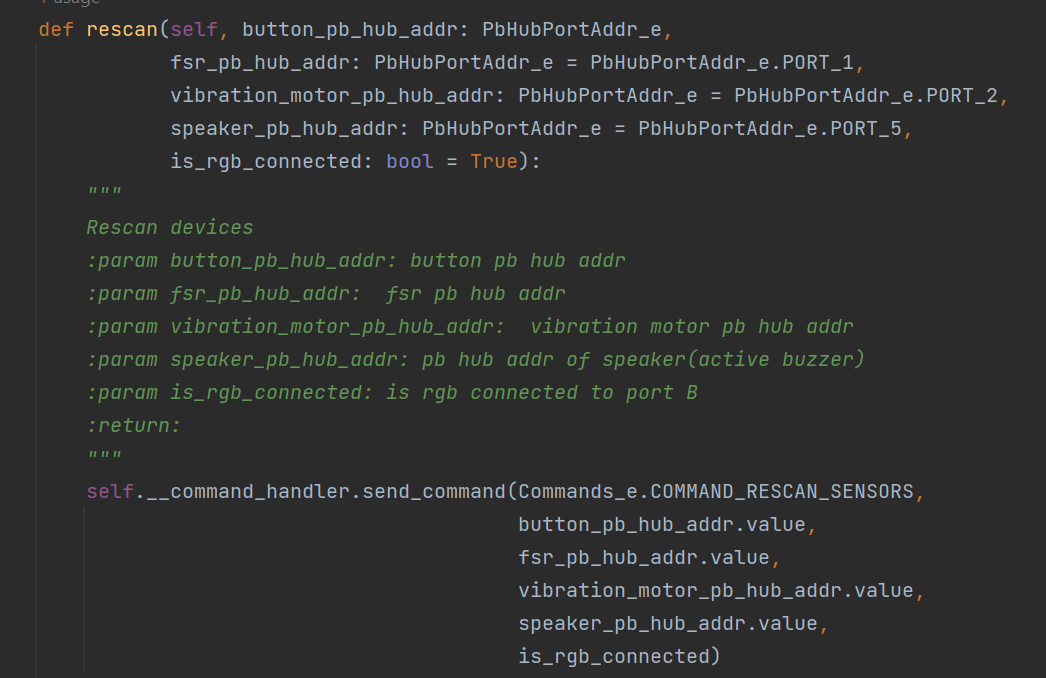
Add command name into 2 Commands\_e enum(master+slave). **Note that enums must be aligned in values otherwise it may lead to unexpected behavior!**

###### Command enum in python

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

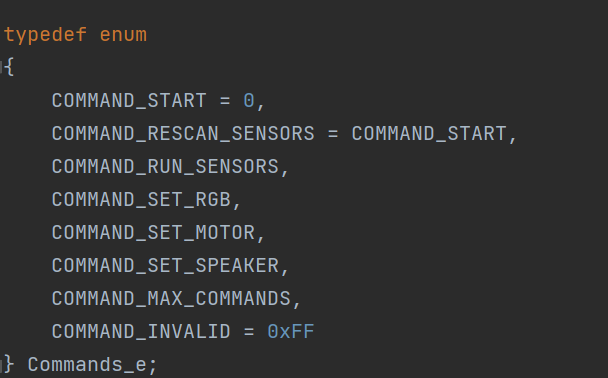


Afterwards create in M5Telemetry the command with specified arguments and pass to send\_command CommandHandler method the command name and the specified arguments:



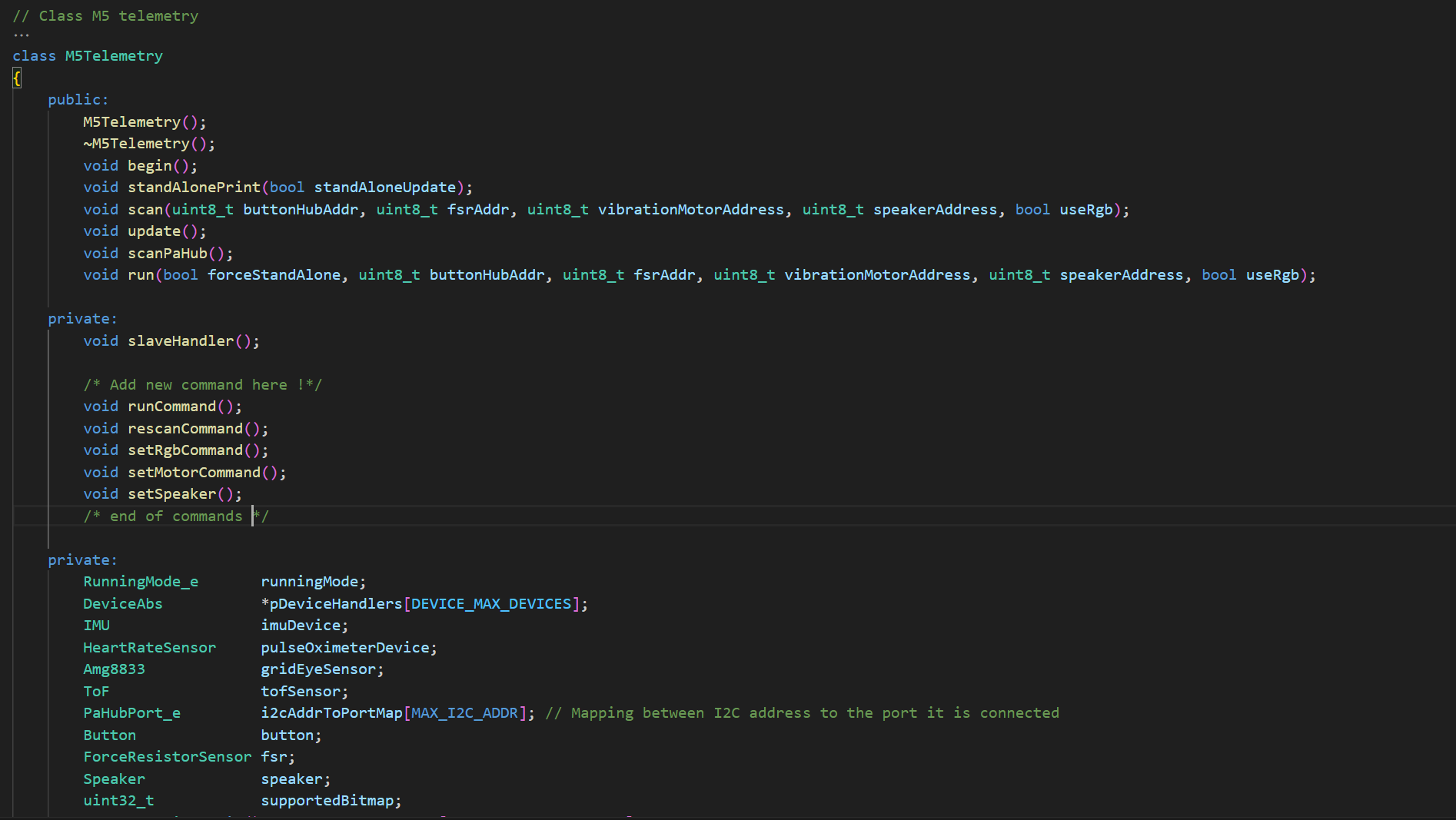
Rescan device example

Slave Commands\_e location: main/SharedDefines.h

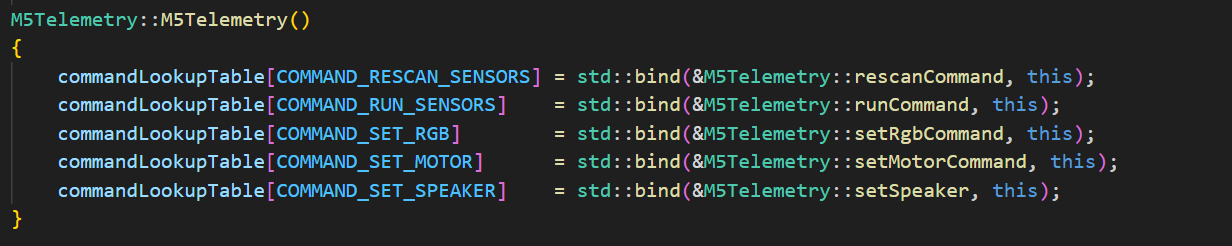


**Note that command must be placed in order before MAX\_COMMANDS!**

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.

### How to add new sensor in FW and Python

#### FW(Arduino)

#### DeviceName\_e

typedef enum

{

    DEVICE\_START\_ALL\_DEVICES = 0,

    DEVICE\_START\_INTERNAL = DEVICE\_START\_ALL\_DEVICES, // Starting point for internal devices

    DEVICE\_IMU = DEVICE\_START\_INTERNAL,  // Internal IMU device

    DEVICE\_END\_INTERNAL,                 // Endpoint for internal devices

    DEVICE\_START\_EXTERNAL\_PA\_HUB = DEVICE\_END\_INTERNAL,  // Starting point for external devices connected to PA HUB

    DEVICE\_HEART\_UNIT\_MAX\_30100 = DEVICE\_START\_EXTERNAL\_PA\_HUB, // External heart unit device

    DEVICE\_AMG8833\_GRID\_EYE,  // External AMG8833 Grid Eye sensor

    DEVICE\_VL3L5CX\_TOF,       // External VL3L5CX TOF sensor

    DEVICE\_END\_EXTERNAL\_PA\_HUB,          // Endpoint for external devices on PA HUB

    DEVICE\_START\_EXTERNAL\_PB\_HUB = DEVICE\_END\_EXTERNAL\_PA\_HUB,  // Starting point for devices on PB HUB

    DEVICE\_FSR402 = DEVICE\_START\_EXTERNAL\_PB\_HUB,  // External Force Sensitive Resistor sensor

    DEVICE\_FSR402\_1,

    DEVICE\_END\_EXTERNAL\_PB\_HUB,          // Endpoint for external devices on PB HUB

    DEVICE\_START\_EXTERNAL\_PORT\_B = DEVICE\_END\_EXTERNAL\_PB\_HUB,  // Starting point for devices on Port B

    DEVICE\_END\_EXTERNAL\_PORT\_B = DEVICE\_START\_EXTERNAL\_PORT\_B,  // Endpoint for external devices on Port B

    DEVICE\_MAX\_DEVICES = DEVICE\_END\_EXTERNAL\_PORT\_B  // Maximum number of supported devices

} DeviceName\_e;

There are 3 connection types in enum:

PaHub, PbHub and port B type.

Add to the enum the new device name.

**Don’t add a member after DEVICE\_MAX\_DEVICES!**

#### Class

The second step is to create a class which inherit from the relevant device type. There are 2 class type the can be inherited:

* PaHubDevicesAbs
* DeviceAbs ( Call PbHub built-in class / Define manually port in PORT B)

An example of ToF:

class ToF: public PaHubDeviceAbs

And speaker:

class Speaker: public DeviceAbs

#### Class cpp file

In case of PaHub device need to pass I2C address, one may note in the constructor of ToF:

 \*/

ToF::ToF() :

imageWidth(0),

imageResolution(0),

maxRes(TOF\_MAX\_ARRAY\_SIZE),

PaHubDeviceAbs(VL3L5CX\_TOF\_I2C\_ADDR)

{

}

Note in PbHub device hubAddr argument must be passed at the begin method (which will be used later on the access the register for R/W transactions in PbHub), Example in speaker:

\*\*

 \* @brief Initializes the Button with a specific address.

 \* @param addr The address to set for the button.

 \* @return true since the operation is always successful.

 \*/

bool Speaker::begin(uint8\_t addr)

{

    hubAddr = addr;

    return true;

}

In case of PbHub Device it is enough to Include "PbHubDevice.h" and use the built-in function or add later-on additional features in PbHub Native API. An example in Speaker

        PbHub.setDigitalWrite0(hubAddr, HIGH);

        delay(1);

        PbHub.setDigitalWrite0(hubAddr, LOW);

        delay(1);

#### M5Telemetry

In main telemetry API there's several steps needed to be done.

First in M5Telemetry header file add a member of your sensor(=Class).

If is it an I2C device (Port A) add after the following section in same structure your new member in list:

    //PA\_HUB handling (I2C)

    pDeviceHandlers[DEVICE\_HEART\_UNIT\_MAX\_30100] = static\_cast<DeviceAbs\*>(&pulseOximeterDevice);

    pDeviceHandlers[DEVICE\_AMG8833\_GRID\_EYE]     = static\_cast<DeviceAbs\*>(&gridEyeSensor);

    pDeviceHandlers[DEVICE\_VL3L5CX\_TOF]          = static\_cast<DeviceAbs\*>(&tofSensor);

pDeviceHandlers[NameInDevice\_eEnum] = static\_cast<DeviceAbs\*>(&ptrToTheMember)

if is it a member of PbHub / Port B need to add an argument of PbHub PbHub Address or a bool flag which indicates if is it connected in Port B. If PbHub address not use please in parameter ( Such example exists in main.ino) set arugment to :

PB\_HUB\_PORT\_INVALID\_ADDR

And for Port B device to false.

**Need to add the argument to scan and run methods.**

In scan method need in case of PbHub device ask if we set a valid PbHubAddress.

In case of sensor:

    if(PB\_HUB\_PORT\_INVALID\_ADDR != fsr1addr)

    {

        pDeviceHandlers[DEVICE\_FSR402\_1] = static\_cast<DeviceAbs\*>(&fsr1);

        fsr1.begin(fsr1addr, DEVICE\_FSR402\_1);

        M5.Lcd.println("FSR1 initialized");

    }

In case of output device such as speaker:

 if (PB\_HUB\_PORT\_INVALID\_ADDR != vibrationMotorAddress)

    {

        M5.Lcd.println("Vibration motor initialized");

        vibrationMotor.begin(vibrationMotorAddress);

    }

And afterwards use the begin of your method.

In case of port B:

    // Port B outputs

    if (useRgb)

    {

        M5.Lcd.println("RGB initialized");

        // PORT B

        RGBDevice.begin();

    }

In case it is sensor as in PbHub/PaHub you need set the member in list of devices.

#### Python

#### Devices

Need to create device class in <REPO>/CLI/Devices

\*In case of sensor need to create set method:

def set(self, data:bytes)

(Which is overridden from DeviceAbs). The method is used to parse the sensor results from FW, therefore it is user response to handle in **Python side** the parsing of the data. As an example in case of ToF we parse it in following manner:

 def set(self, data: bytes):

        # Update mm\_distances as before

        self.mm\_distances = np.array(struct.unpack(f"<{ToF.tof\_num\_of\_pixels}H", data)).reshape(8, 8)

        # Avoid zero distances to prevent issues with the geometric mean

        self.temp\_mm\_distances = np.where(self.mm\_distances == 0, 1e-12, self.mm\_distances)

        # Append the current matrix to the history

        self.mm\_distances\_history.append(self.temp\_mm\_distances)

        # Keep only the last 'history\_length' matrices

        while len(self.mm\_distances\_history) > ToF.history\_length:

            self.mm\_distances\_history.pop(0)

        # Compute the average matrix

        self.mm\_distances\_avg = np.mean(self.mm\_distances\_history, axis=0)

#### CommandHandler

Need to add in <REPO>/CLI/Devices/DeviceAbs.py

There's an enum of all sensors Device\_e. You need to add your sensor to the enum. **Note Enum values must be aligned with FW Device Enum !**

#### M5Telemtry

In <REPO>/CLI/M5Telemetry.py

At the constructor, add your need device as a member of class and add it the the dictionary of devices:

        # Initialize devices.

        self.fsr = Fsr(Device\_e.FSR)

        self.fsr1 = Fsr(Device\_e.FSR1)

        self.imu = Imu()

        self.tof = ToF()

        self.amg = Amg8833()

        self.hru = HRU()

        # Mapping of Device enums to device instances for easy updating of device values.

        self.devices = {

            Device\_e.FSR: self.fsr,

            Device\_e.FSR1: self.fsr1,

            Device\_e.IMU: self.imu,

            Device\_e.TOF: self.tof,

            Device\_e.AMG833: self.amg,

            Device\_e.HRU: self.hru

        }

* In case of output sensor a new command need to be created(There's an explanation in Slave section of how to add command)

## Working with PI

* Working with PI – supports only WIFI mode.
* For any related instructions / tutorial for usage RaspberryPI please use README in Raspberry PI directory in repository.



# 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 doesn’t lights up correctly (Issue with the Power consumption)
* Unreliable data transmission to raspberry pi via serial connection.