M5Telemetry

M5StackCore3 Telemetry Infrastructure

Yonatan Amir and Yuri Lukach

Index

1 ..... M5Telemetry
3 ..... Background
4 .....Equipment
13 ..... Equipment Testing
13 ..... Planning
14 ..... Results
28 .....Software
28 .....Installation
32 ..... State machine
32 .....Arduino
32 .....Python(Master)
34 .....Software Design
34 ..... M5Telemetry
35 .....Block Diagram
35 .....Schematics
36 .....SharedDefines.h
36 ..... Usage
37 .....Slave
41 .....Python(Master)
44 .....How to add new command
47 .....How to add new sensor in FW and Python
51 .....Working with PI
52 .....Warnings
52 ..... Bugs

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.

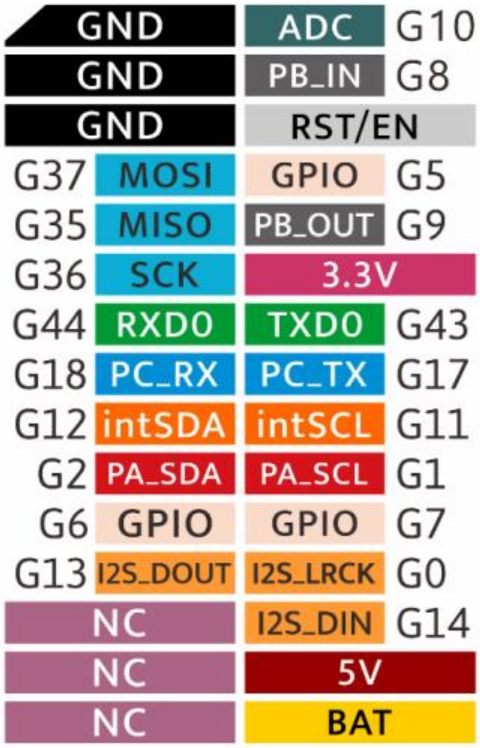
One of the standout features of the M5Stack is its flexibility in accommodating a variable number of sensors. Unlike more rigid systems that necessitate a fixed quantity of sensors, the M5Stack allows us to easily incorporate and adapt to a diverse array of sensors. This adaptability empowers students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

Resources	Parameters
MCU	ESP32-S3@Xtensa LX7, 16MFLASH AND 8M-PSRAM, WIFI, OTGCDC functions
Touch the IPS LCD screen	2.0"@320*240 ILI9342C
Camera	GC0308@30 megapixels
Proximity sensors	LTR-553ALS-WA
Power management chip	AXP2101
Six-axis attitude sensor	BMI270
magnetometer	BMM150
RTC	BM8563
Speaker	16bits I2S power amplifier chip AW88298@TW
Audio decoding chip	ES7210, dual microphone inputs
Product Size	54 x 54 x 16mm
Package Size	101x64x34mm
Product Weight	73.3g
Package Weight	97.8g

Pinout



LCD Pixel:320x240 TF cards support up to 16GB

ESP32S3 Chip	GPIO35	GPIO37	GPIO36	GPIO3	GPIO15	AXP2101_DO1
ILI9342C	MISO	MOSI	SCK	CS	RST	BL

ESP32S3 Chip	GPIO35	GPIO37	GPIO36	GPIO4
TF Card	MISO	MOSI	SCK	CS

Camera

ESP32S3 Chip	GPIO12	GPIO11	AW9523B_P1_0	GPIO45	GPIO46	GPIO38
GC0308	SDA	SCL	CAM_RST	CAM_PCLK	CAM_VSYNC	CAM_HREF

CAP.TOUCH (I2C Addr: 0x58)

ESP32S3 chip	GPIO12	GPIO11	AW9523B_P1_2	AW9523B_P0_0
FT6336U	SDA	SCL	INT	RST

AXP Power Led

AXP2101	AXP_CHGLED
Red LED	Vcc

RTC

ESP32S3 Chip	GPIO12	GPIO11	AXP2101_WAKEUP
BM8563	SDA	SCL	INT

IMU (3-axis gyroscope + 3-axis accelerometer)

ESP32S3 Chip	GPIO12	GPIO11
BMI270&BMM150	SDA	SCL

Internal I2C connection

ESP32S3 Chip	GPIO12	GPIO11
BMI270&BMM150	SDA	SCL
AXP2101	SDA	SCL
BM8563	SDA	SCL
ES7210	SDA	SCL
AW88298	SDA	SCL

- 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.

REG MAP (Addr:0x61)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	note
function / channel	IO0 Digital Write (1 byte) (r/w)	IO1 Digital Write (1 byte) (r/w)	IO0 PWM (1 byte) (r/w)	IO1 PWM (1 byte) (r/w)	IO0 Digital Read (1 byte) (r)	IO1 Digital Read (1 byte) (r)	IO0 Analog Read (2 bytes) (r/w)[2]	reserve	Set RGB LED Numbers (2 bytes) (r/w) (default:74[1])	Set one LED Color (5 bytes) (r/w)	Set more LED Color (7 bytes) (r/w)	Set RGB LED Brightness (1 bytes) (r/w)	IO0 Servo Angle (1 byte) (r/w)[3]	IO1 Servo Angle (1 byte) (r/w)	IO0 Servo Pulse (2 byte) (r/w)	IO1 Servo Pulse (2 byte) (r/w)	The STM32 version gpio output and input voltage is 3.3V
ch0 cmd	0x40	0x41	0x42	0x43	0x44	0x45	0x46	0x47	0x48	0x49	0x4A	0x4B	0x4C	0x4D	0x4E	0x4F	
ch1 cmd	0x50	0x51	0x52	0x53	0x54	0x55	0x56	0x57	0x58	0x59	0x5A	0x5B	0x5C	0x5D	0x5E	0x5F	
ch2 cmd	0x60	0x61	0x62	0x63	0x64	0x65	0x66	0x67	0x68	0x69	0x6A	0x6B	0x6C	0x6D	0x6E	0x6F	
ch3 cmd	0x70	0x71	0x72	0x73	0x74	0x75	0x76	0x77	0x78	0x79	0x7A	0x7B	0x7C	0x7D	0x7E	0x7F	
ch4 cmd	0x80	0x81	0x82	0x83	0x84	0x85	0x86	0x87	0x88	0x89	0x8A	0x8B	0x8C	0x8D	0x8E	0x8F	
ch5 cmd	0xA0	0xA1	0xA2	0xA3	0xA4	0xA5	0xA6	0xA7	0xA8	0xA9	0xAA	0xAB	0xAC	0xAD	0xAE	0xAF	
0xFE: Firmware Version(1 byte)(r)[4]																	
0xFF: Change I2C Address(1 byte)(r/w)[5]																	
[1] STM32 version: 74 is the max of RGB LED numbers																	
[2] STM32 version: ADC width 12bit(0–4095); ATMEGA328 version: ADC width 10bit(0–1023)																	
[3] Servo angle and servo pulse is the new function of the STM32 version. Servo angle: 0–180, Servo pulse: 500–2500																	
[4] This is the new function of the STM32 version, it will return the firmware version																	
[5] This is the new function of the STM32 version, you can set the I2C address from 1 to 127 or get the current I2C address																	

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.

Parameter	Technical data
Digital resolution	Accelerometer (A): 16-bit or 0.06 mg/LSB Gyroscope (G): 16-bit or 0.004 dps/LSB
Programmable measurement range & sensitivity	(A): ±2 g: 16384 LSB/g to (A): ±16 g: 2048 LSB/g (G): ±125 dps: 262.1 LSB/dps to (G): ±2000 dps: 16.4 LSB/dps
Zero-g/zero-rate offset	(A): ±20 mg (G): ±0.5 dps
Sensitivity error	(A): ± 0.4% (G): ± 0.4% (with CRT)
Temperature range	-40 ... +85 °C
Temperature behaviour (TCO; TCS)	(A): ±0.25 mg/K; ±0.004 %/K (G): ±0.02 dps/K; ±0.02 %/K
Noise density	(A): 160 µg/√Hz (G): 0.008 dps/√Hz
Offset vs PCB strain	(A): ±0.01 mg/µε (G): ±1.5 mdps/µε
Filter BW (programmable)	(A): 5 Hz ... 684 Hz (G): 11 Hz ... 751 Hz
Output data rate (ODR)	(A): 12.5 Hz ... 1.6 kHz (G): 25 Hz ... 6.4 kHz
Digital inputs/outputs	2x SPI; 2x I2C; AUX I/F; OIS I/F 2x digital interrupts
Supply voltage	1.7 ... 3.6 VDD 1.2 ... 3.6 VDDIO
Current consumption	685 µA at full ODR (aliasing-free)
Package size	2.5 x 3.0 x 0.8 mm³ 14 pin LGA

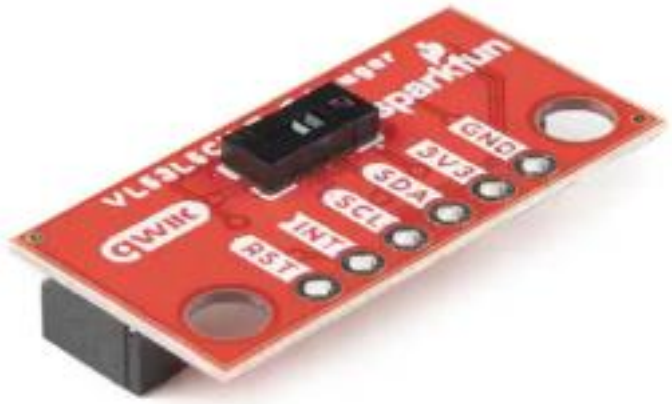
Proximity Sensor

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.



Feature	Details
Package	Optical LGA16
Size	6.4 x 3.0 x 1.5 mm
Ranging	2 to 400 cm per zone
Operating voltage	IOVDD: 1.8 or 2.8 V or 3.3 V AVDD: 2.8 V or 3.3 V
Operating temperature	-30 to 85°C
Sample rate	Up to 60 Hz
Infrared emitter	940 nm
I^2C interface	I^2C: 400 kHz to 1 MHz serial bus, address: 0x52
Operating ranging mode	Continuous or Autonomous (see UM2884 for more information)

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

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.

Specification

Resources	Parameter
Net weight	11g
Gross weight	25g
Product Size	48*24*12mm
Package Size	61*56*17mm

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

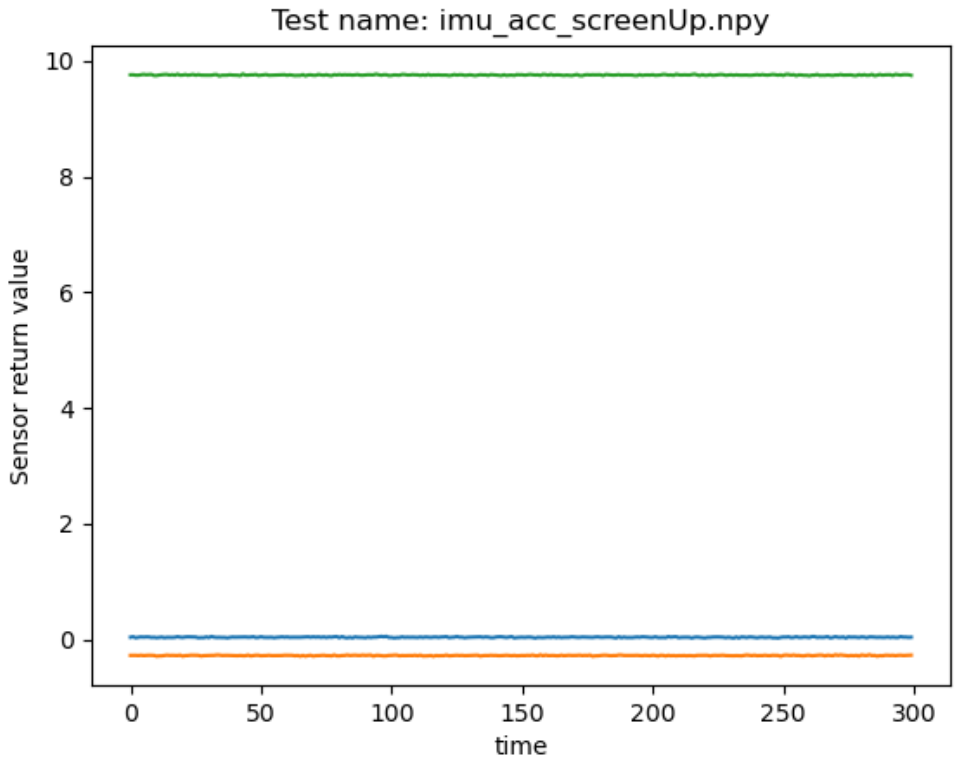
IMU (Inertial Measurement Unit)

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.

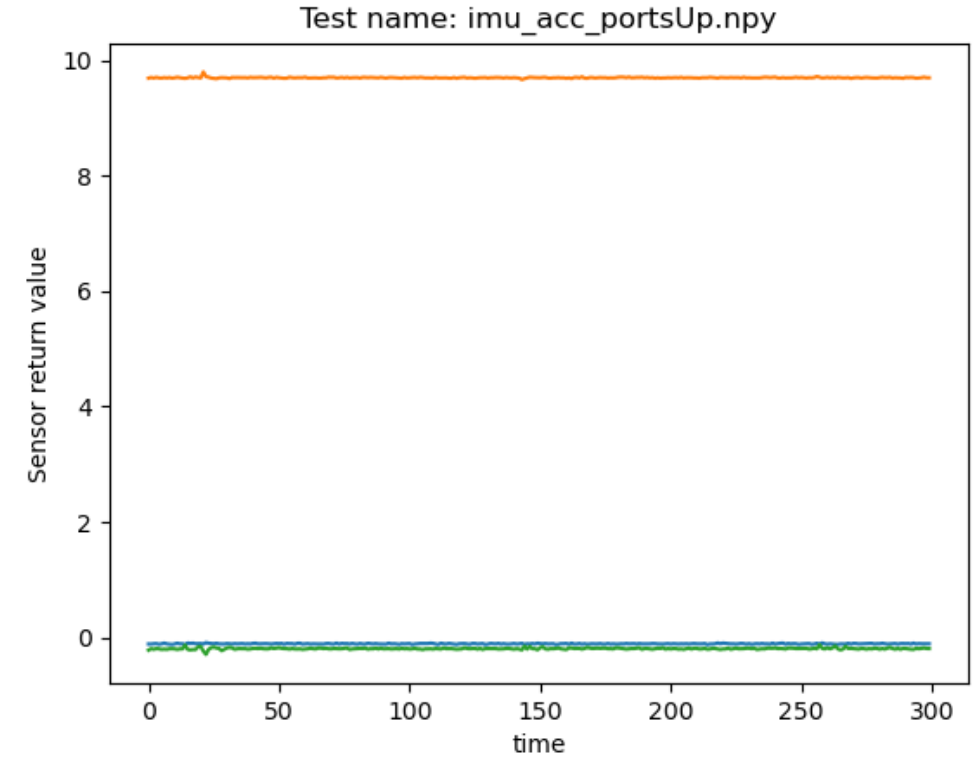
In the following graphs, the [X,Y,Z] vector corresponds to [Blue, Orange, Green].

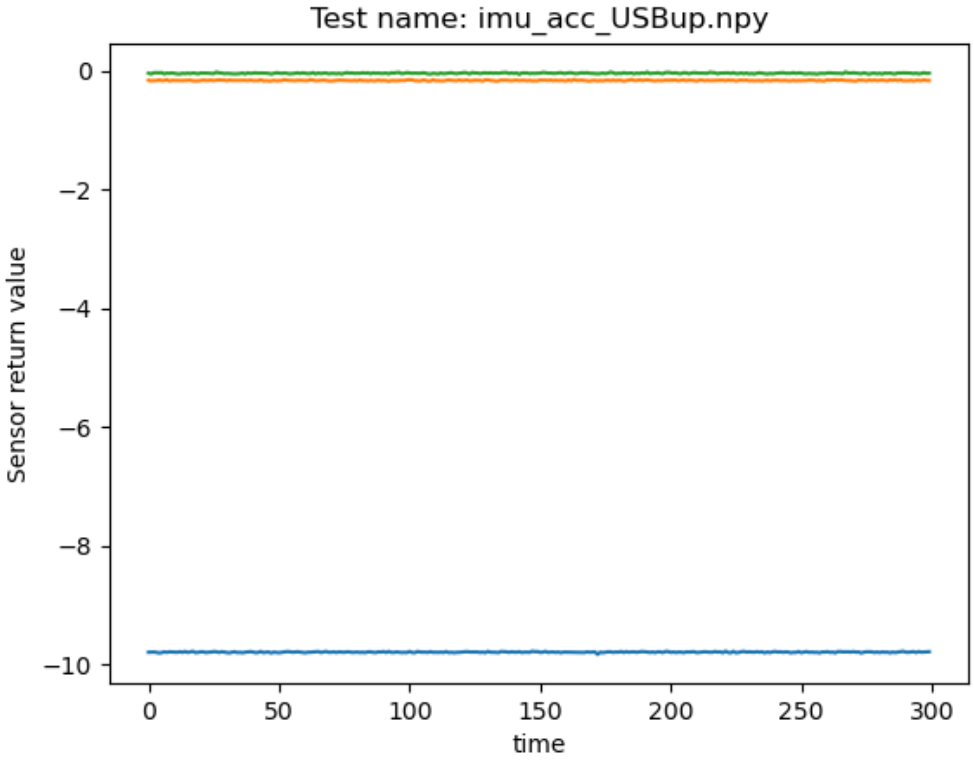
Static Testing

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.


Test name: imu_acc_portsUp.npy
Sensor return value
time
10
8
6
4
2
0
0
50
100
150
200
250
300


Test name: imu_acc_USBup.npy
Sensor return value
time
0
−2
−4
−6
−8
−10
0
50
100
150
200
250
300

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.

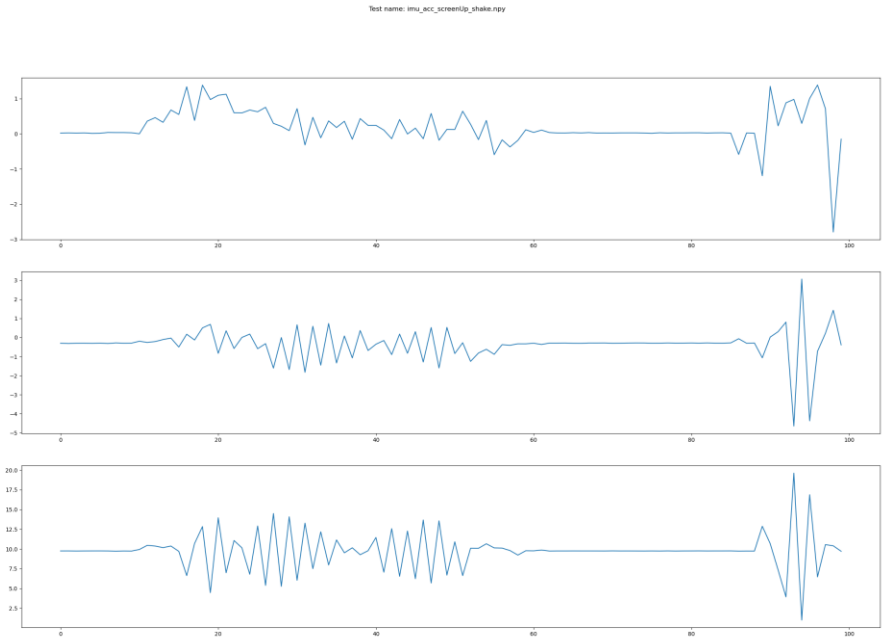
The USB down position is positive in X Axis.

The IIC port up position is positive in Y axis.

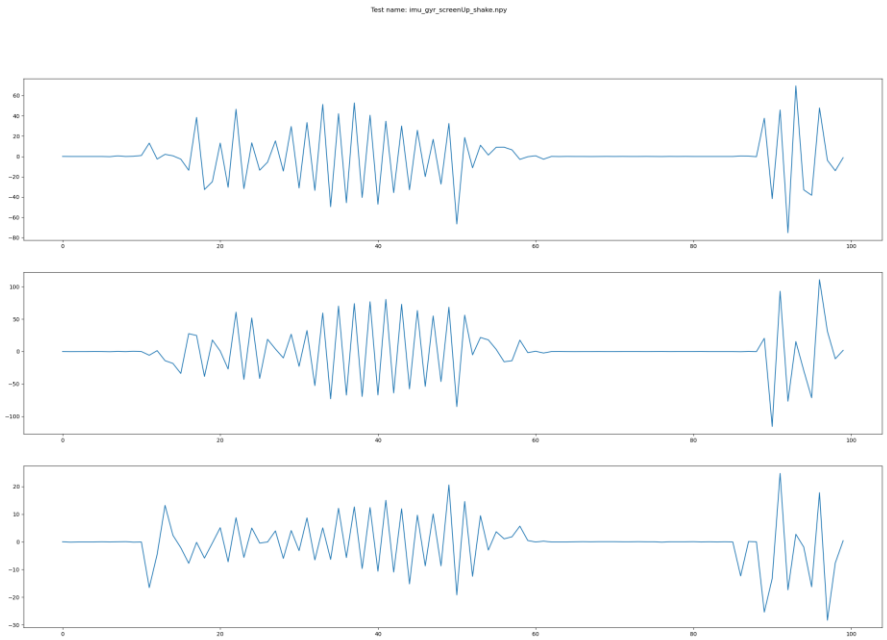
Dynamic Tests

In the dynamic tests, we did 3 tests.

1. Shake test, screen up.



This figure shows results of the accelerometer.

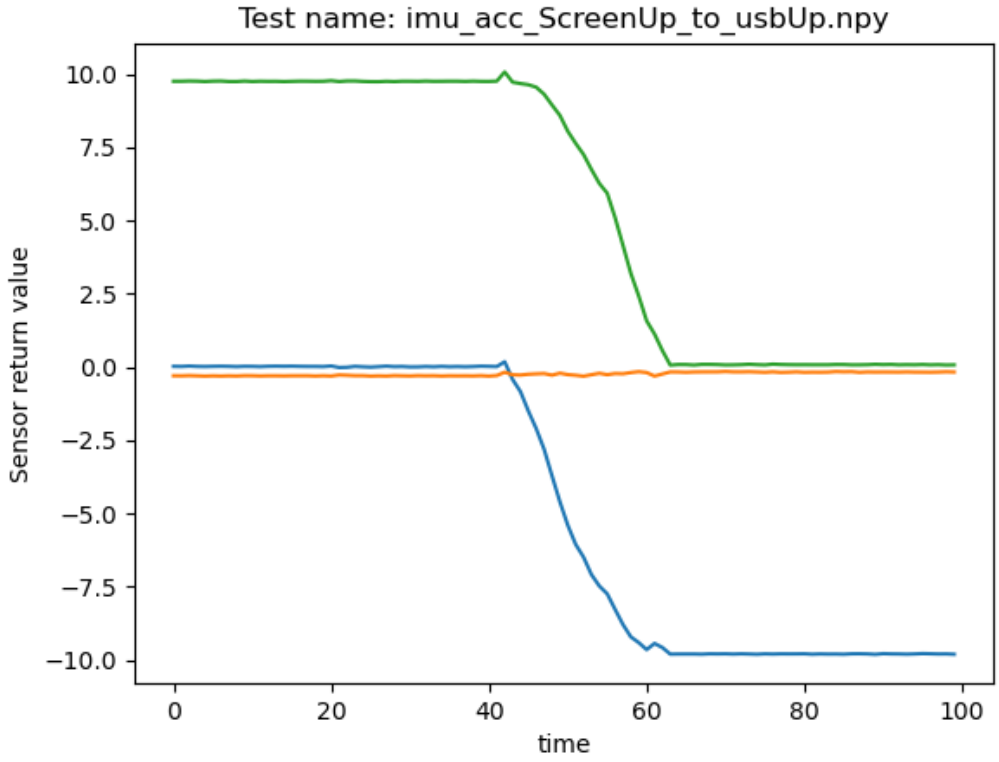


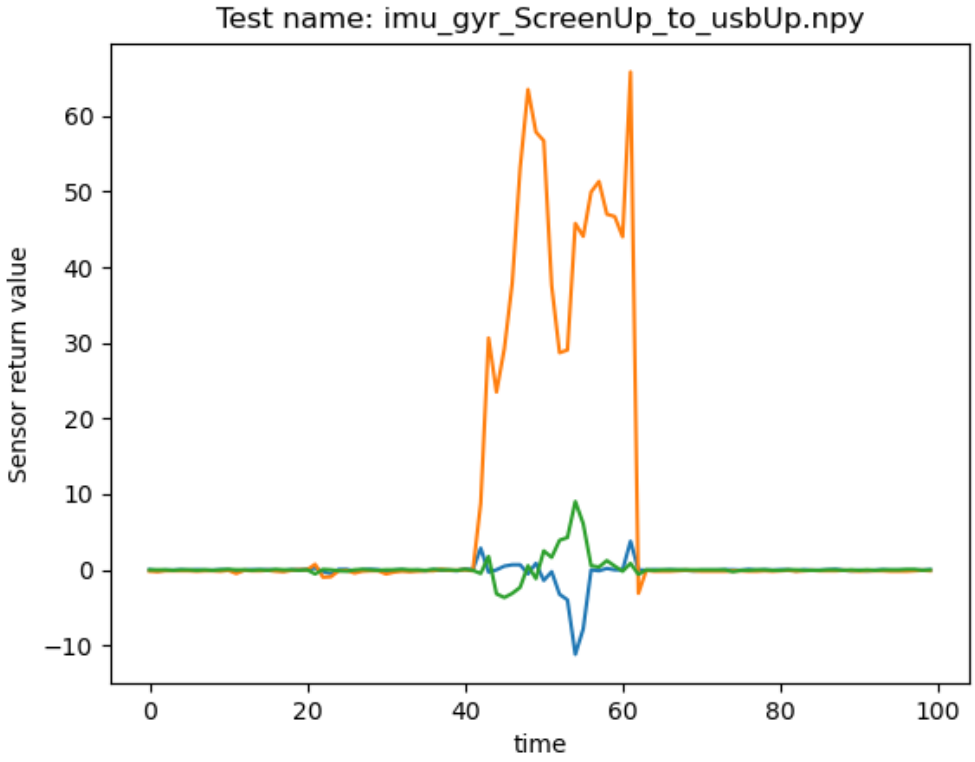
This figure shows the results of the gyroscope.

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.

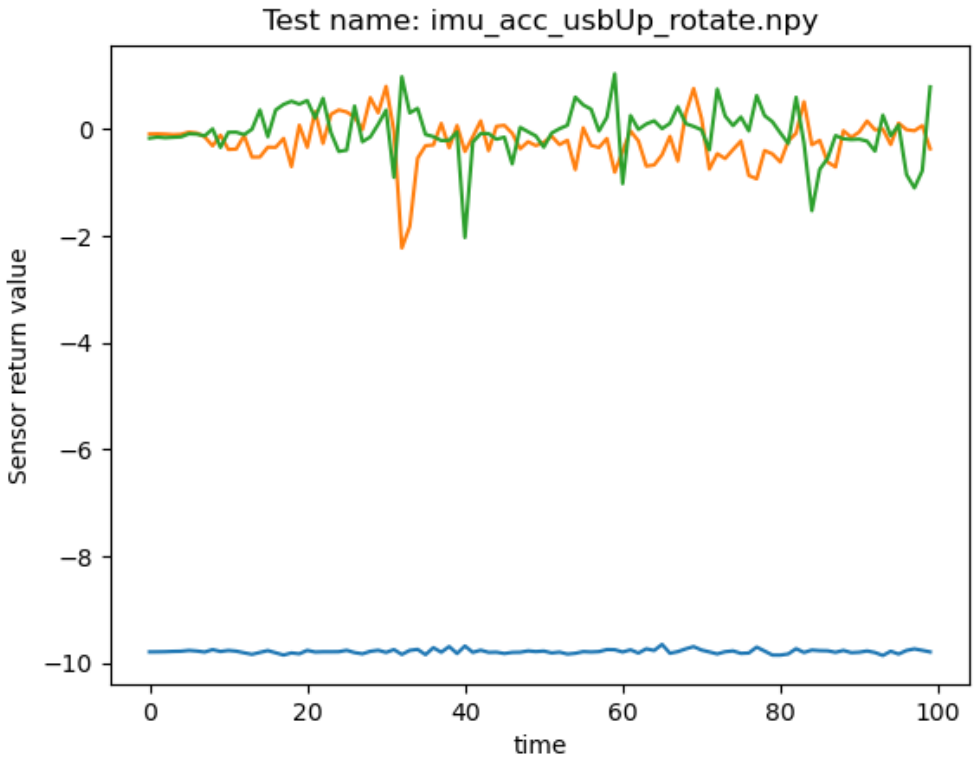


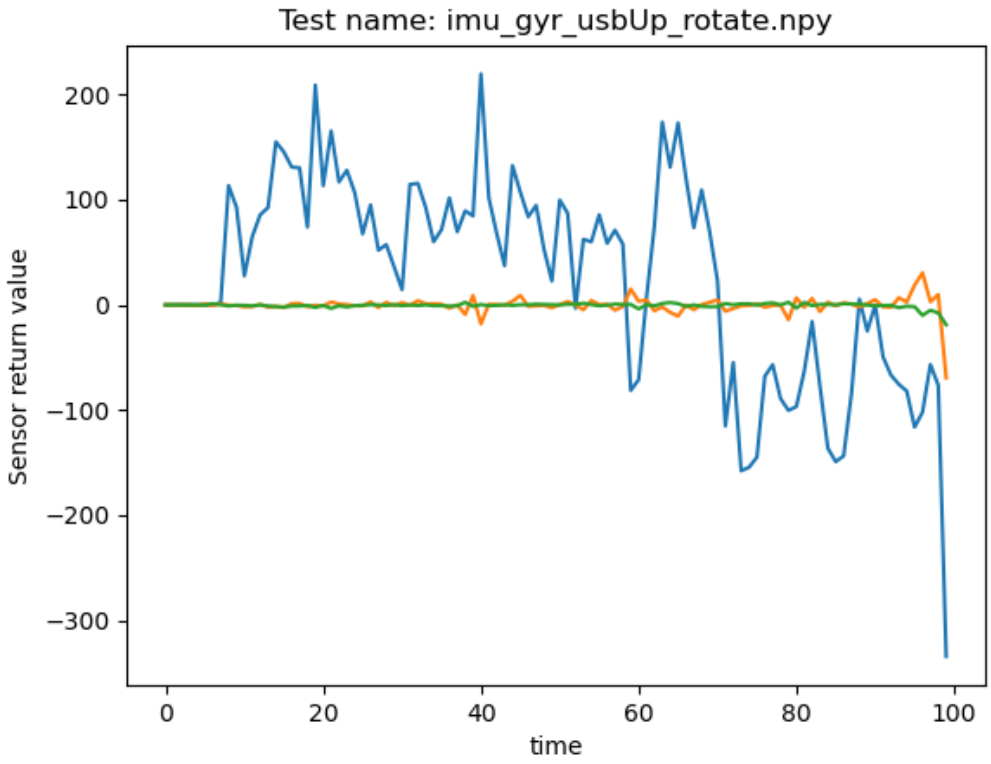


As we can see, both the accelerometer and the gyroscope respond to the movement.

3. USB up rotation

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





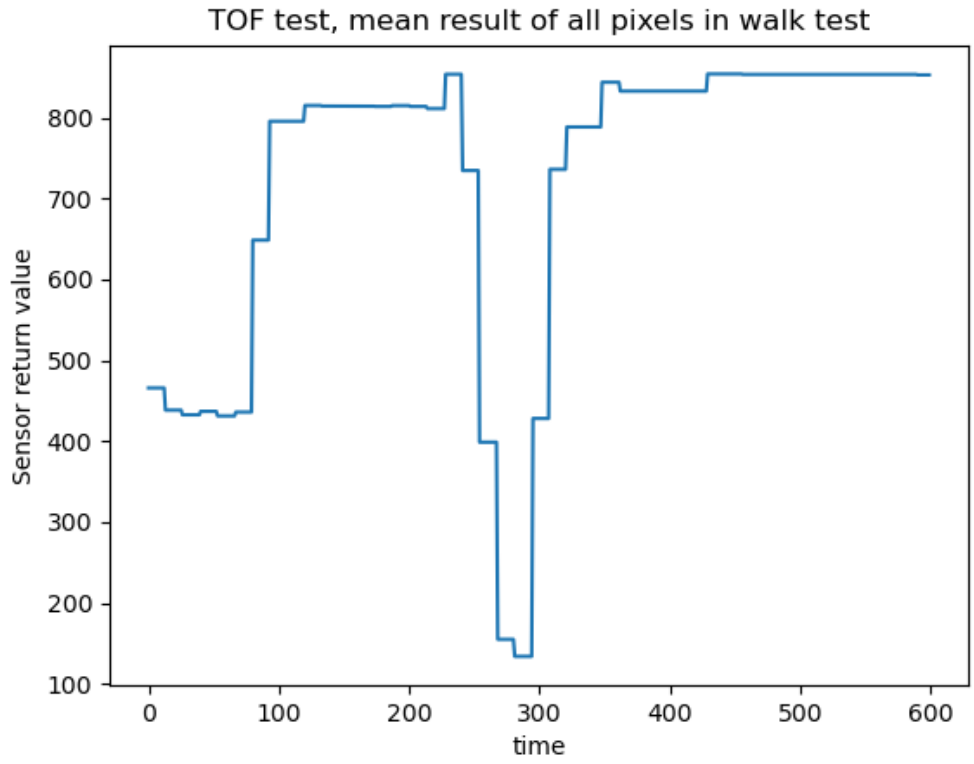
In this case, only the gyroscope shows a response in only one axis, as expected.

Proximity Sensor (TOF) -

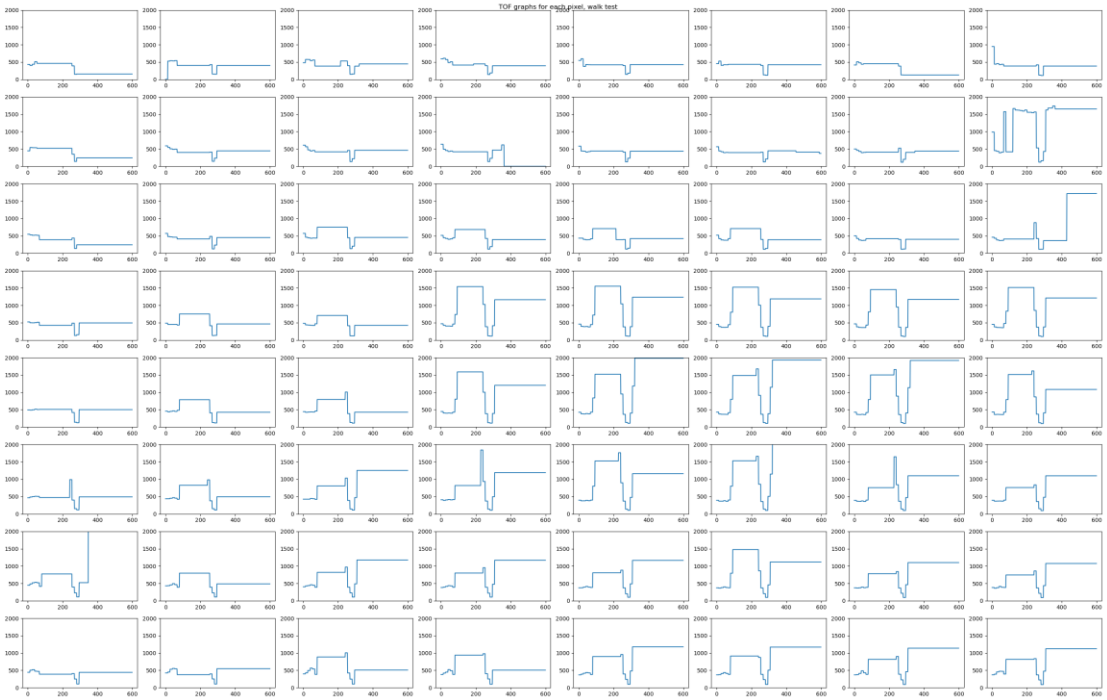
The tests were done in oversampling to also test the sample rate of the sensor.

Walk test

The walk test yields the following results.



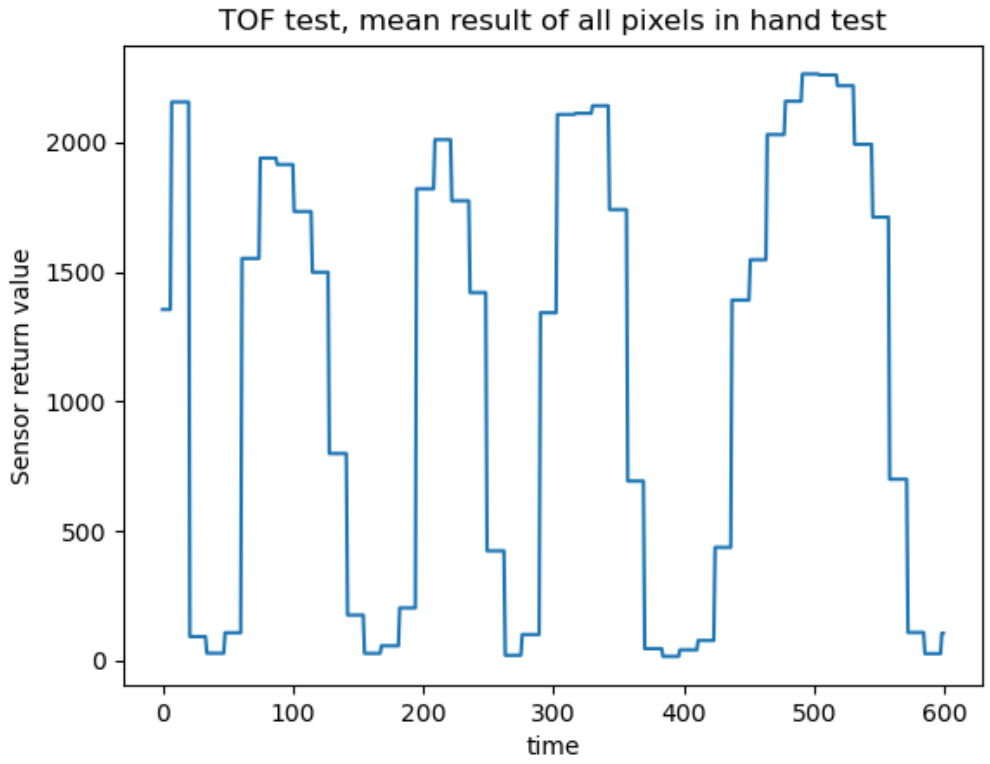
The figure displays the average across all pixels in order to show clearly the increase and decrease in distance. We should note that while a far away body should give high results, a far away body is not seen in all pixels, so the average returned value is not indicative of actual distance. Such use 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

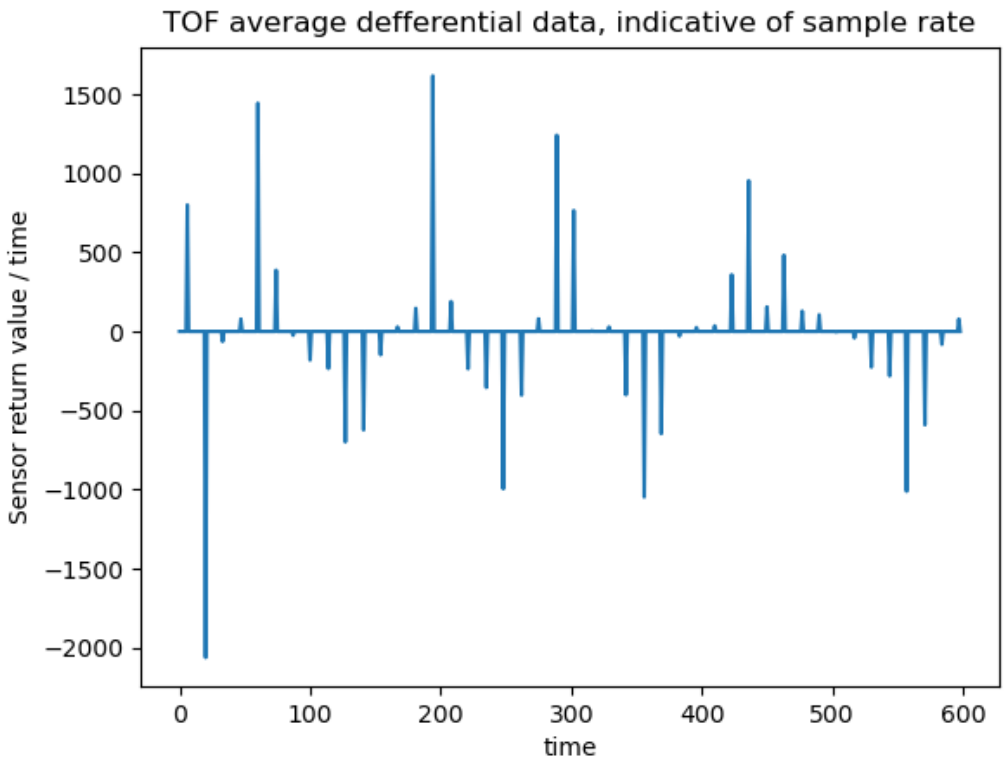
The hand wave results is shown below.



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.

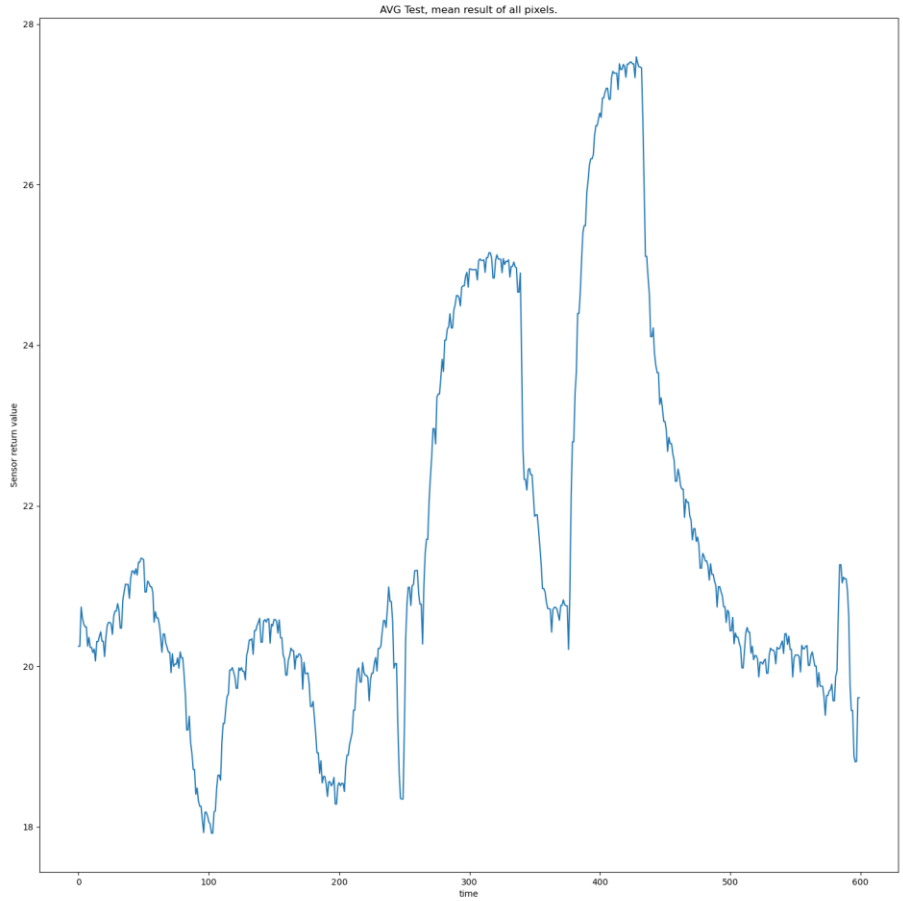
We shall now examine the derivative of this graph.



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.

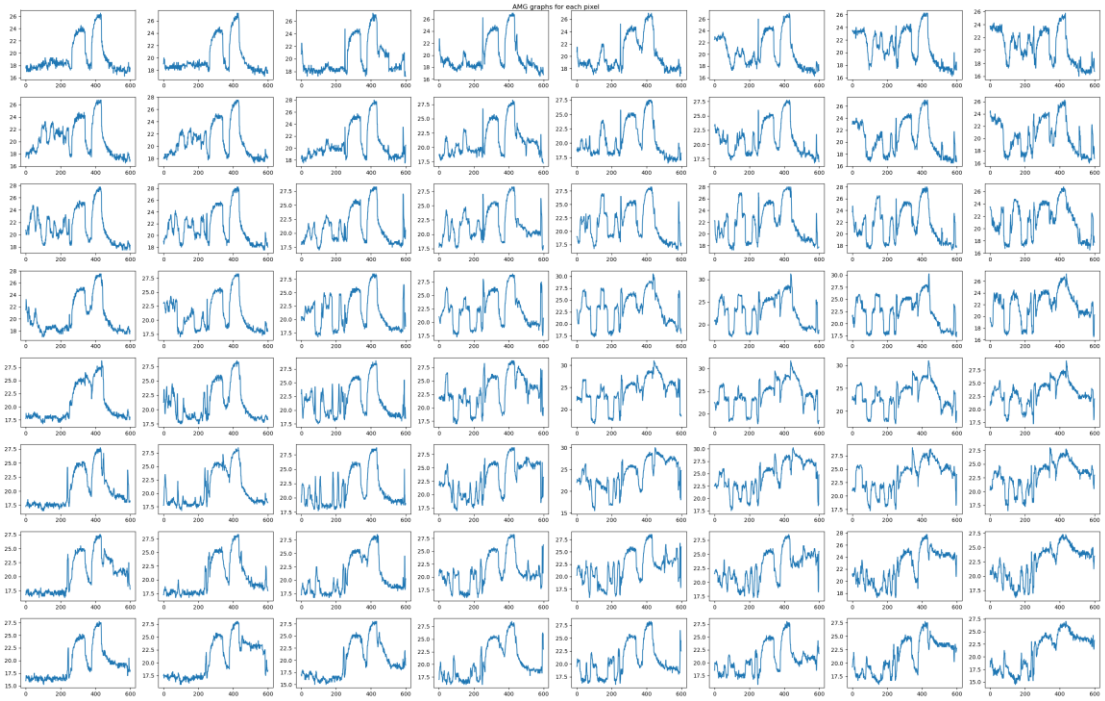
Thermal Imaging unit (AMG) -

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.

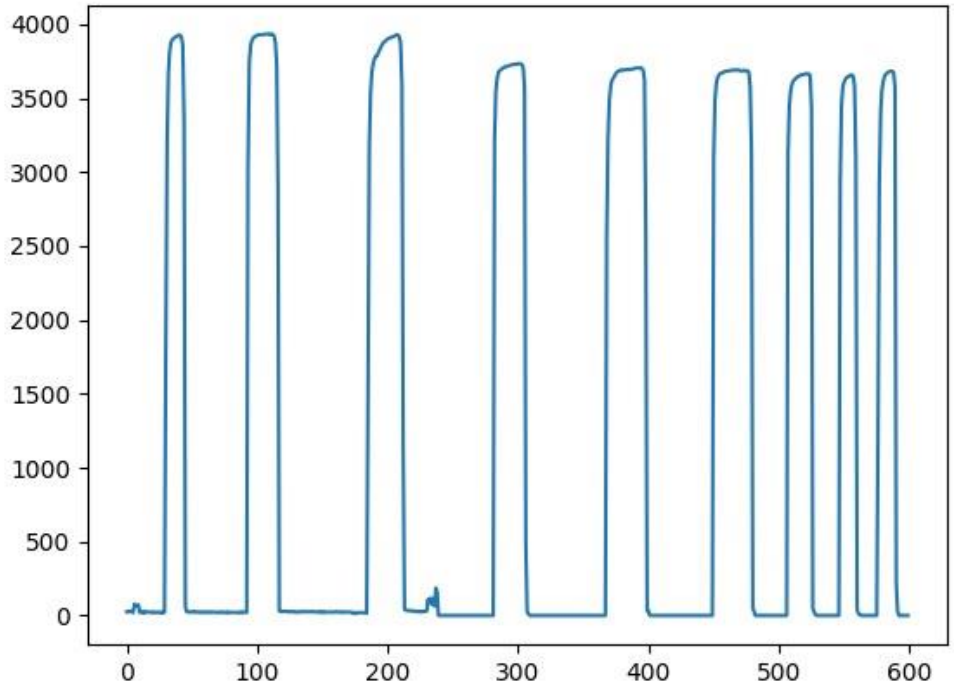


This sensor is useful in it's ability to help with detecting dangerously hot surfaces or items, or in helping determine whether a camera is filming a person or a photo of a person.

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

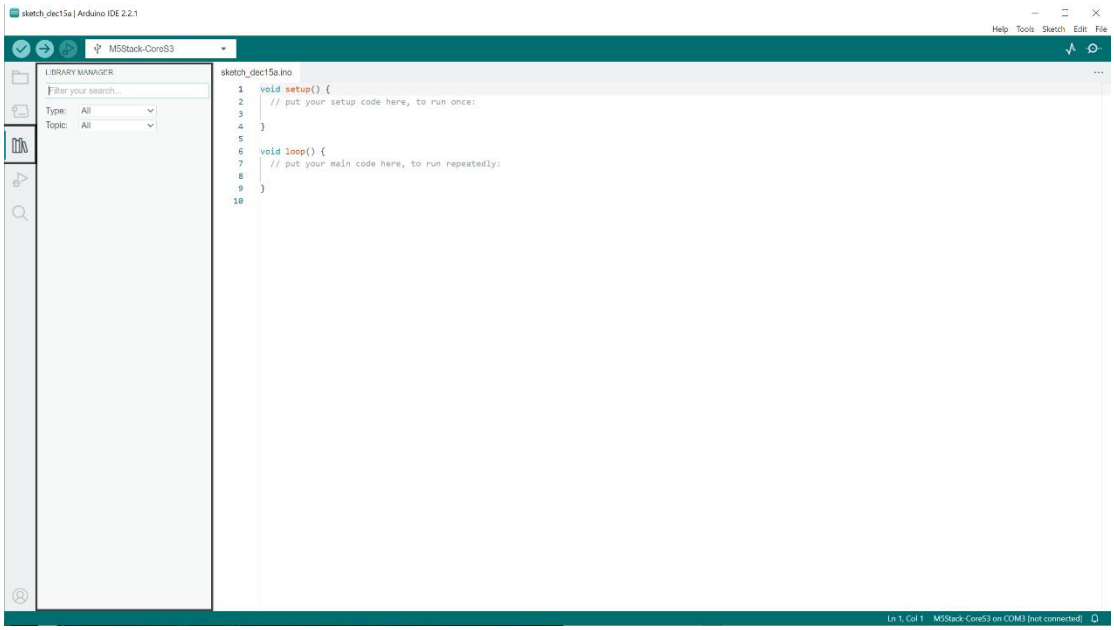
Arduino IDE

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

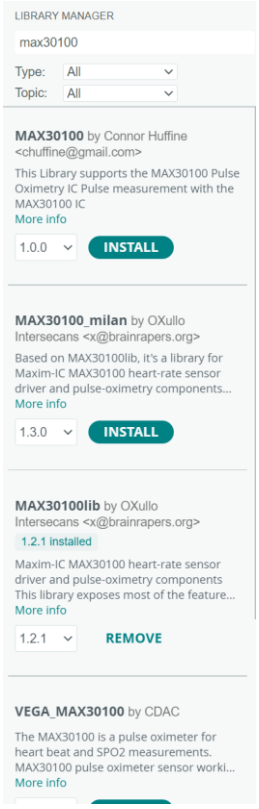
The link below is a tutorial installation provided by M5CORE team regard supporting M5Core in Arduino IDE:

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.

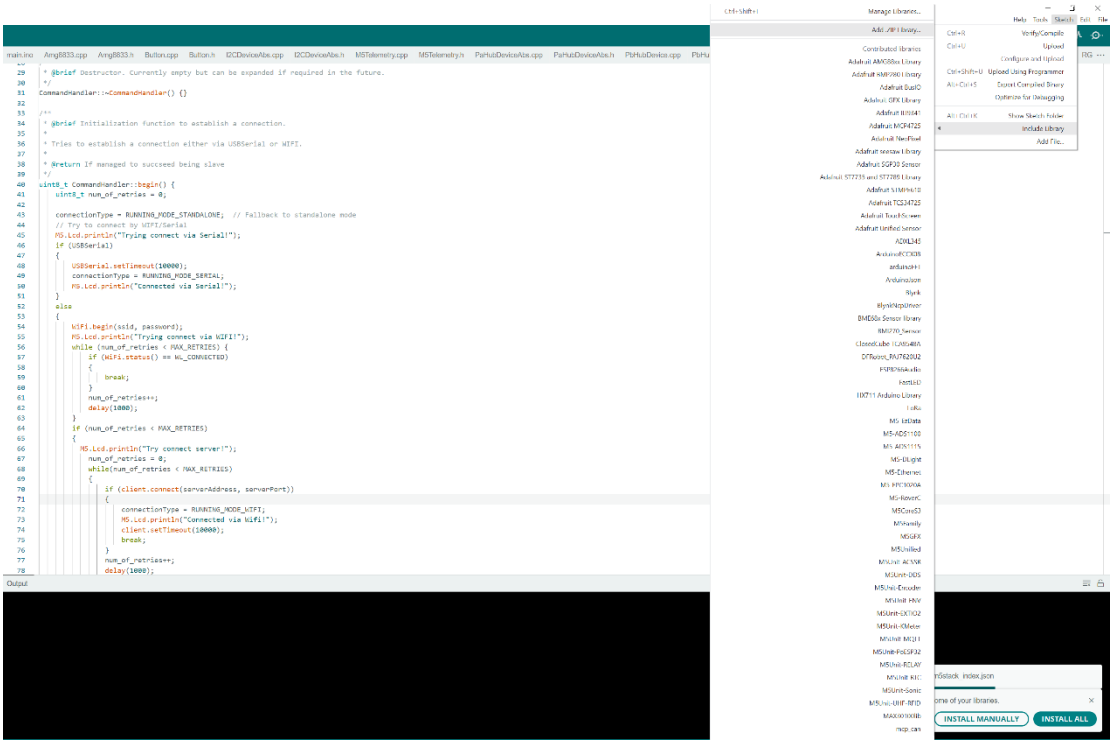
Libraries required:

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

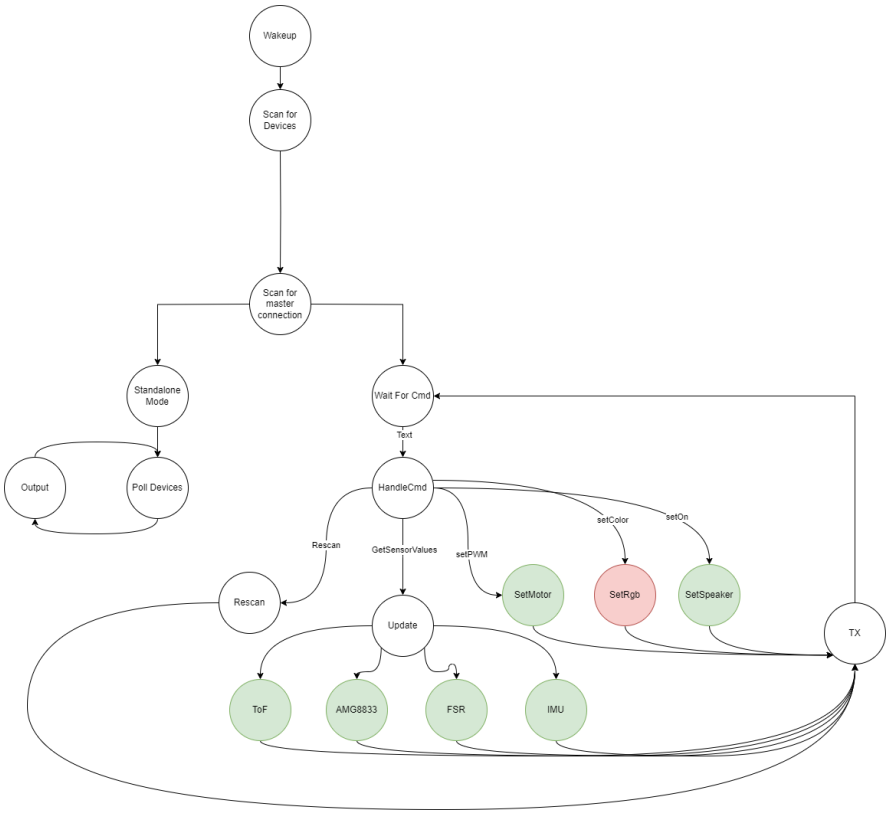
Arduino

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.

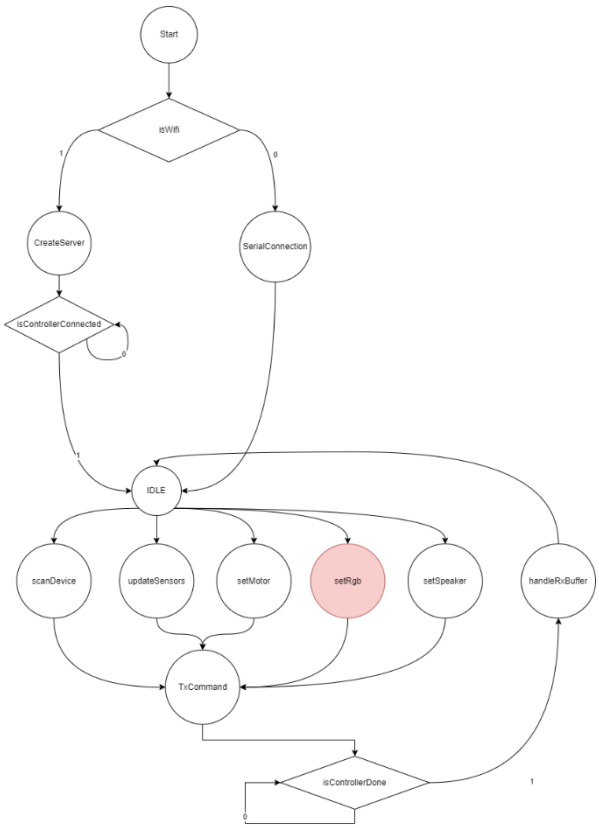


Python(Master)

It can be seen that at the beginning, it is checked whether the user in prison requested to create a connection via Wifi or Serial. If it is a Wifi connection,

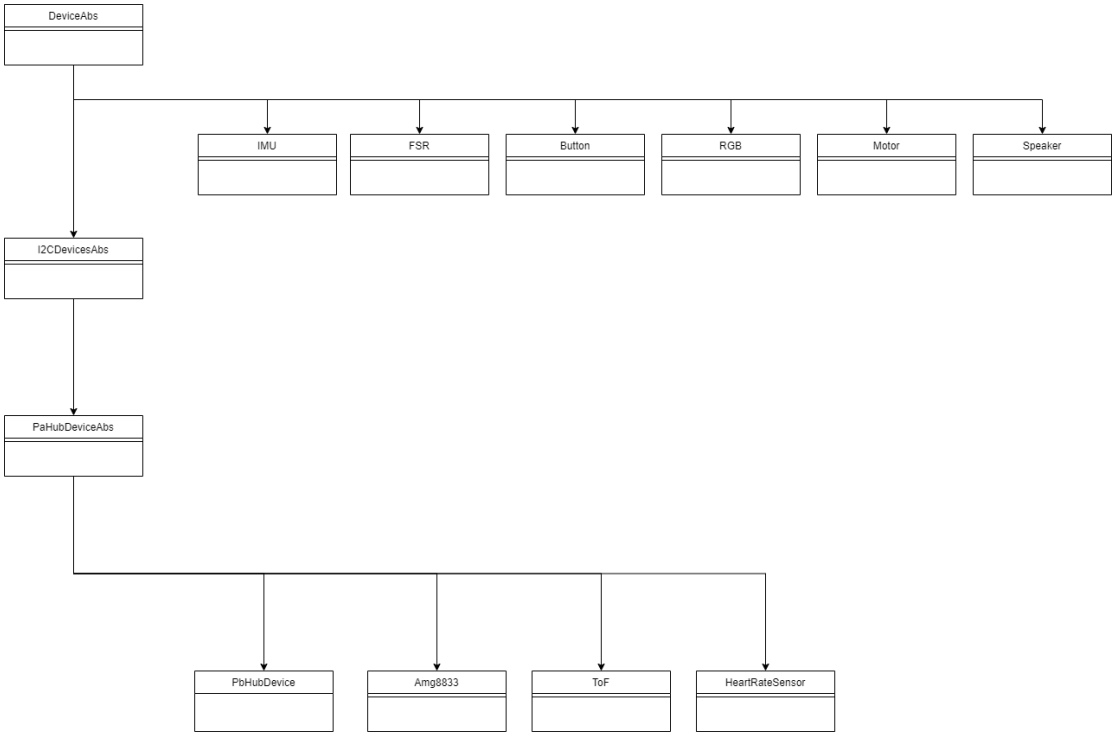
the master creates a server and waits for the controller to connect. If it is a serial connection – Python creates a serial connection against the controller.

After the connection, Python is in IDLE mode, and waits for a command that the user will enter, that same command it broadcasts to the controller, and the controller will handle it. Upon completion of the command, the controller will notify the master whether there is information to be transferred back, if so the master will wait for the broadcast of all the information from the controller. Then, if there is information received from the controller, Python will handle the request according to the command, if there is no information the buffer (which contains all the information) will be empty and will return to IDLE mode again



Software Design

Device



M5Telemetry

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

Arduino basics

Arduino

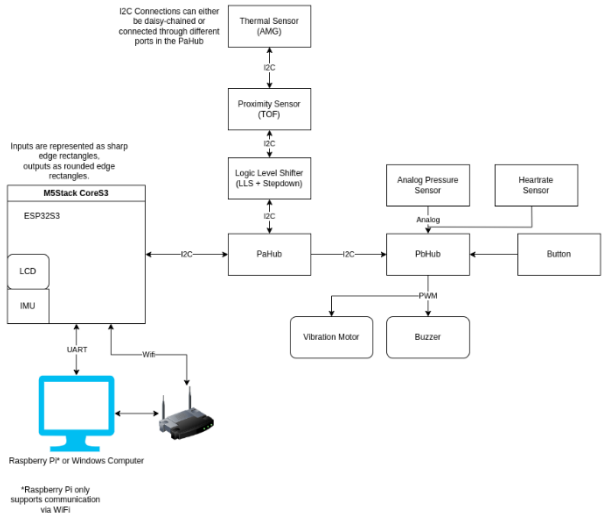
Arduino code is quite straightforward and typically consists of two main phases:

- 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.

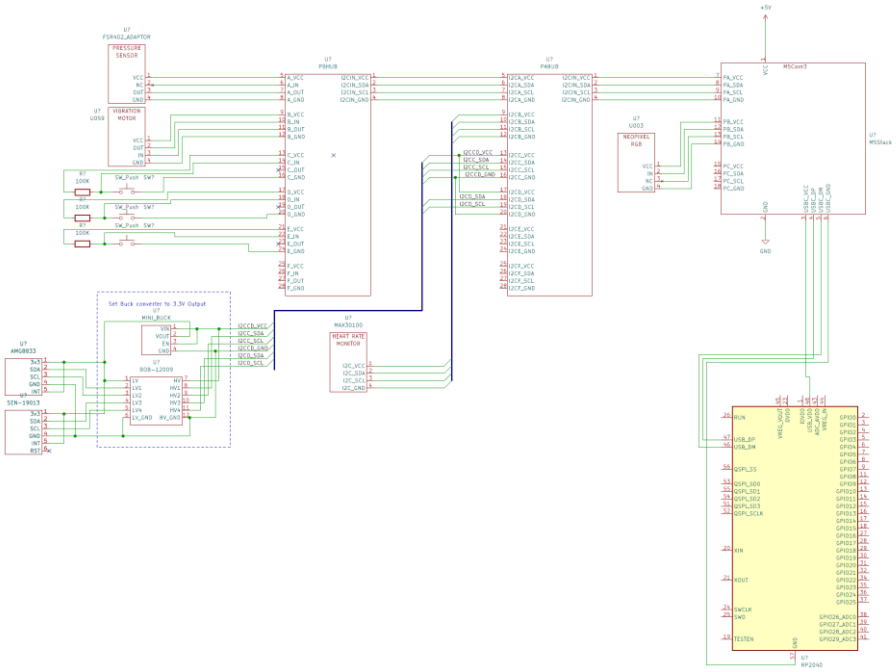
```ino
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

learning the basics of Arduino programming, you can review helpful tutorials on websites like Tutorialspoint's Arduino section.

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:

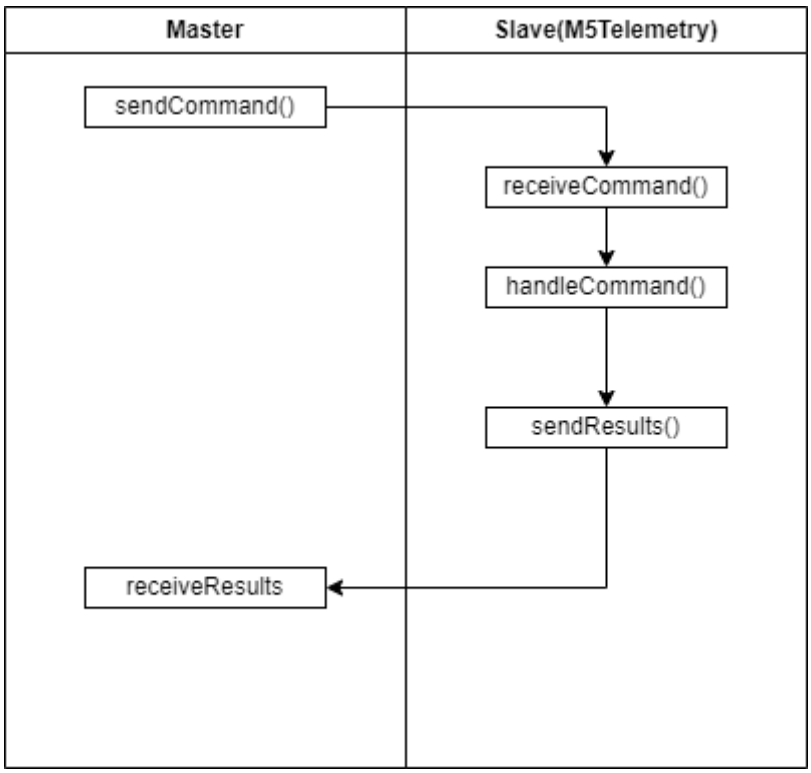
```
#include "M5Telemetry.h"
#include "SharedDefines.h"

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

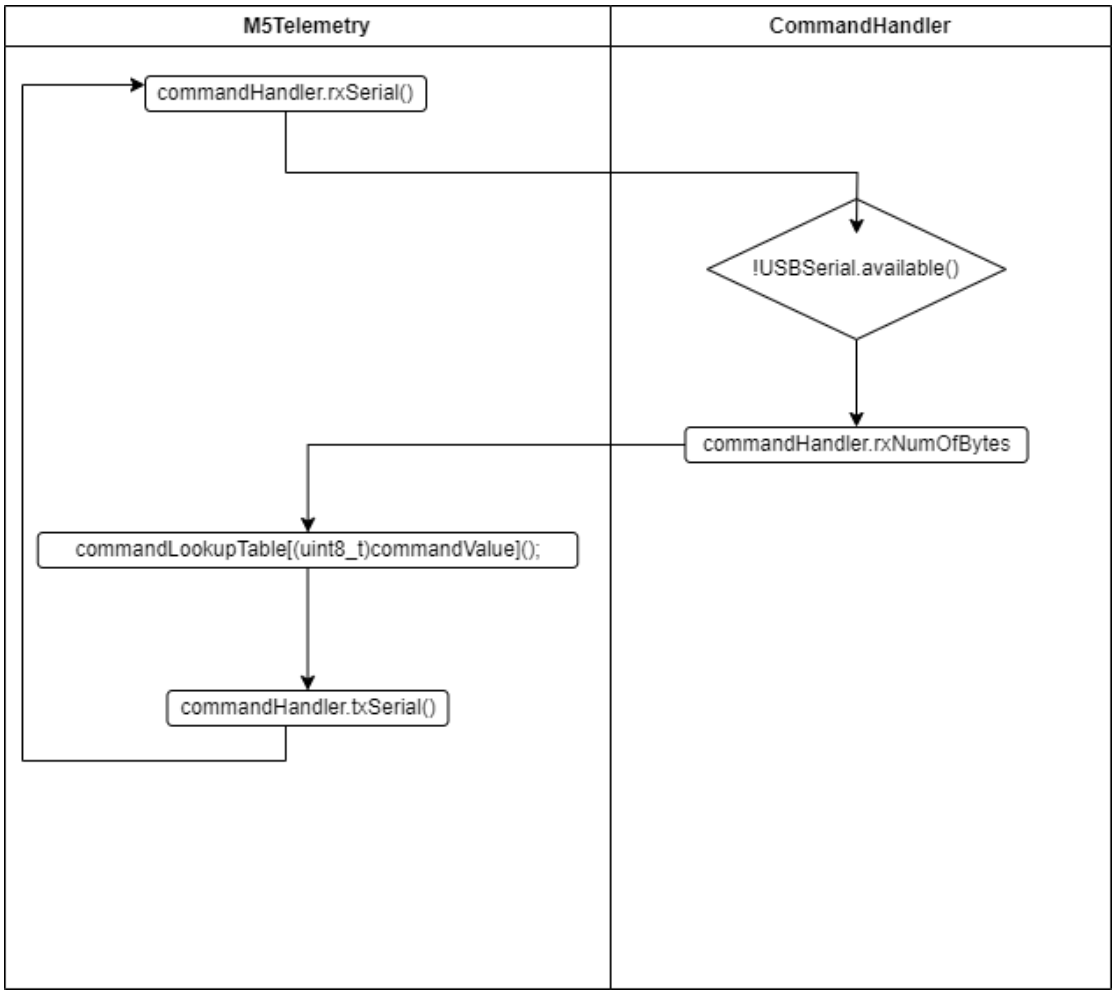


There are two buffers used for command handling:

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.

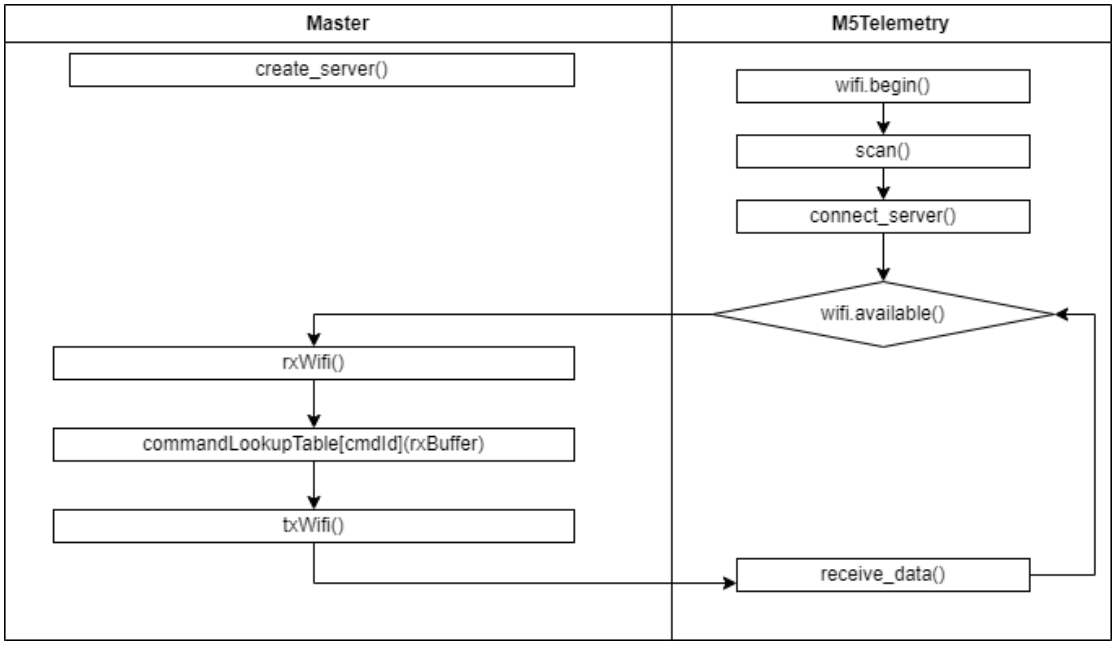
Just ensure when connecting via serial connection that Arduino actually recognize it.

```
#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:

```
// WIFI Hardcoded data
const char* ssid = "Free-TAU";
const char* password = "free-tau";
const char* serverAddress = "172.30.6.174";
const int serverPort = 12345;
```

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

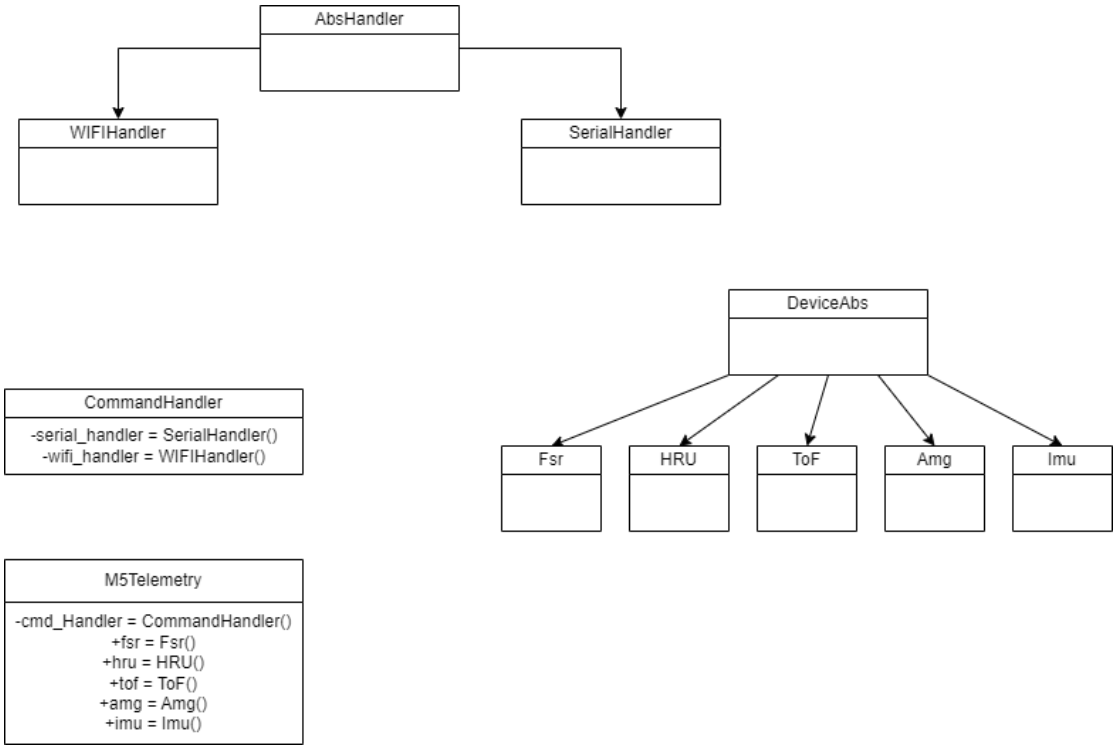
```
class WifiHandler(AbsHandler):

    host = "0.0.0.0"
    port = 12345
```

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
```

```python
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:

```
FSR value: 0
Imu(Gyro=GyroData(x=   0, y=  -0, z=  -0), Accel=AccelData(x=   0, y=  -0, z=  10))
AMG88xx ° 8x8:
[[26.2 26.5 26.8 26.2 26.2 26.2 26.5 27.5]
 [26.5 26.2 26.8 26.5 26.2 25.8 26.8 27.0]
 [25.5 26.2 26.2 26.0 26.5 26.5 26.0 26.2]
 [25.8 26.0 25.8 26.5 26.5 26.2 26.0 27.0]
 [25.2 26.0 25.5 26.2 25.8 26.2 26.0 26.5]
 [25.2 26.2 25.8 26.0 26.0 26.0 26.2 25.8]
 [25.2 25.2 25.8 25.8 25.8 26.2 26.2 26.5]
 [25.5 25.2 25.2 25.0 25.8 25.8 26.0 26.5]]
ToF distances[mm] 8x8:
[[1766    6    5    3    4    4    4    4]
 [   2    5    3    4    4    7    5    5]
 [   4    4    0    5    7    6    8    7]
 [   5    4    0    5    4    6    6    7]
 [   4    5    3    4    6    4    5    8]
 [ 447    4    1    5    7    7    7    8]
 [ 430  475    5    3  461  422  438  383]
 [ 367  428    2    4    5  399  373  356]]
set motor with duty cycle of 50
FSR value: 0
```

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 snap-
shots from TOF and IMU
        x, y, z = interface.imu.gyro_data.x, interface.imu.gyro_data.y, in-
terface.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_tem-
prature_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

```
class Commands_e(enum.IntEnum):
    """
    Supported commands. Commands must be aligned Commands_e of main/SharedDefines.h
    """
    COMMAND_RESCAN_SENSORS = 0
    COMMAND_RUN_SENSORS = 1
    COMMAND_SET_RGB = 2
    COMMAND_SET_MOTOR = 3
    COMMAND_SET_SPEAKER = 4
```

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

```
    def rescan(self, button_pb_hub_addr: PbHubPortAddr_e,
               fsr_pb_hub_addr: PbHubPortAddr_e = PbHubPortAddr_e.PORT_1,
               vibration_motor_pb_hub_addr: PbHubPortAddr_e = PbHubPortAddr_e.PORT_2,
               speaker_pb_hub_addr: PbHubPortAddr_e = PbHubPortAddr_e.PORT_5,
               is_rgb_connected: bool = True):
        """
        Rescan devices
        :param button_pb_hub_addr: button pb hub addr
        :param fsr_pb_hub_addr:  fsr pb hub addr
        :param vibration_motor_pb_hub_addr:  vibration motor pb hub addr
        :param speaker_pb_hub_addr: pb hub addr of speaker(active buzzer)
        :param is_rgb_connected: is rgb connected to port B
        :return:
        """
        self.__command_handler.send_command(Commands_e.COMMAND_RESCAN_SENSORS,
                                            button_pb_hub_addr.value,
                                            fsr_pb_hub_addr.value,
                                            vibration_motor_pb_hub_addr.value,
                                            speaker_pb_hub_addr.value,
                                            is_rgb_connected)
```

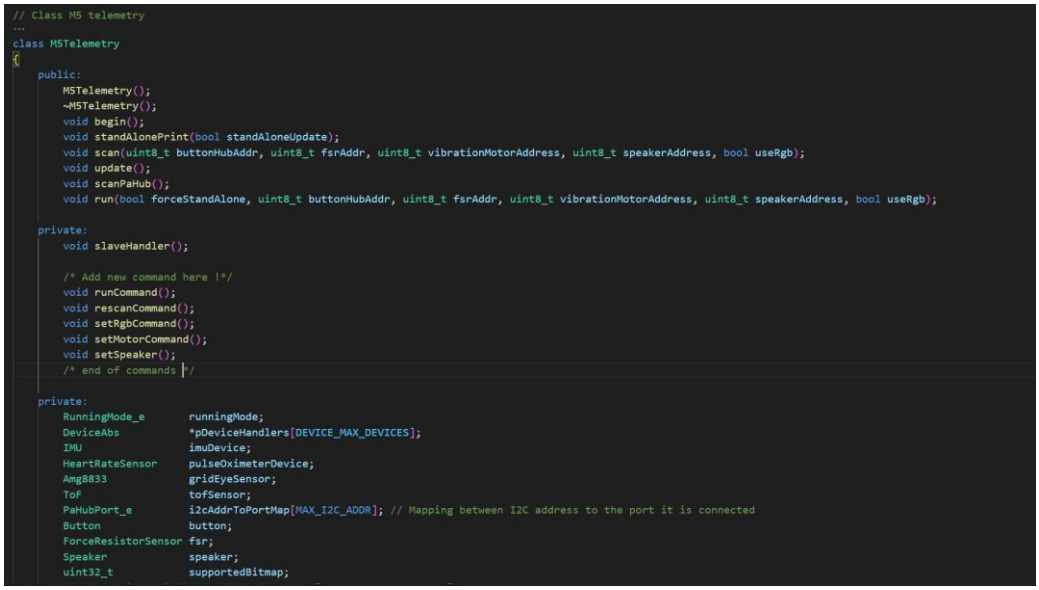
Rescan device example

Slave Commands_e location: main/SharedDefines.h

```
typedef enum
{
    COMMAND_START = 0,
    COMMAND_RESCAN_SENSORS = COMMAND_START,
    COMMAND_RUN_SENSORS,
    COMMAND_SET_RGB,
    COMMAND_SET_MOTOR,
    COMMAND_SET_SPEAKER,
    COMMAND_MAX_COMMANDS,
    COMMAND_INVALID = 0xFF
} Commands_e;
```

Note that command must be placed in order before MAX_COMMANDS!

Create a method in M5Telemetry API between comments:



```
// Class M5 telemetry
...
class M5Telemetry
{
    public:
        M5Telemetry();
        ~M5Telemetry();
        void begin();
        void standAlonePrint(bool standAloneUpdate);
        void scan(uint8_t buttonHubAddr, uint8_t fsrAddr, uint8_t vibrationMotorAddress, uint8_t speakerAddress, bool useRgb);
        void update();
        void scanPaHub();
        void run(bool forceStandAlone, uint8_t buttonHubAddr, uint8_t fsrAddr, uint8_t vibrationMotorAddress, uint8_t speakerAddress, bool useRgb);

    private:
        void slaveHandler();

        /* Add new command here !*/
        void runCommand();
        void rescanCommand();
        void setRgbCommand();
        void setMotorCommand();
        void setSpeaker();
        /* end of commands */

    private:
        RunningMode_e       runningMode;
        DeviceAbs           *pDeviceHandlers[DEVICE_MAX_DEVICES];
        IMU                 imuDevice;
        HeartRateSensor     pulseOximeterDevice;
        Amg8833             gridEyeSensor;
        ToF                 tofSensor;
        PaHubPort_e         i2cAddrToPortMap[MAX_I2C_ADDR]; // Mapping between I2C address to the port it is connected
        Button              button;
        ForceResistorSensor fsr;
        Speaker             speaker;
        uint32_t            supportedBitmap;
```

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:

```
M5Telemetry::M5Telemetry()
{
    commandLookupTable[COMMAND_RESCAN_SENSORS] = std::bind(&M5Telemetry::rescanCommand, this);
    commandLookupTable[COMMAND_RUN_SENSORS]    = std::bind(&M5Telemetry::runCommand, this);
    commandLookupTable[COMMAND_SET_RGB]        = std::bind(&M5Telemetry::setRgbCommand, this);
    commandLookupTable[COMMAND_SET_MOTOR]      = std::bind(&M5Telemetry::setMotorCommand, this);
    commandLookupTable[COMMAND_SET_SPEAKER]    = std::bind(&M5Telemetry::setSpeaker, this);
}
```

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

```
/**
 * @brief Processes the set rgb command.
 * @details set RGB unit
 */
void M5Telemetry::setRgbCommand()
{       You, 3 weeks ago • CommandHandler …
    uint8_t id    = static_cast<uint8_t>(commandHandler.bufferToUint32(RxBuffer + 8));
    uint8_t red   = static_cast<uint8_t>(commandHandler.bufferToUint32(RxBuffer + 12));
    uint8_t green = static_cast<uint8_t>(commandHandler.bufferToUint32(RxBuffer + 16));
    uint8_t blue  = static_cast<uint8_t>(commandHandler.bufferToUint32(RxBuffer + 20));

    RGBDevice.SetRGB(id, red, green, blue);

}
```

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 in-
ternal 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 sup-
ported 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<Device-
Abs*>(&pulseOximeterDevice);
    pDeviceHandlers[DEVICE_AMG8833_GRID_EYE]     = static_cast<Device-
Abs*>(&gridEyeSensor);
    pDeviceHandlers[DEVICE_VL3L5CX_TOF]          = static_cast<Device-
Abs*>(&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_pix-
els}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.