Data collection instruction for a rig with high-speed bldc motor and a drone propeller

August 13, 2025

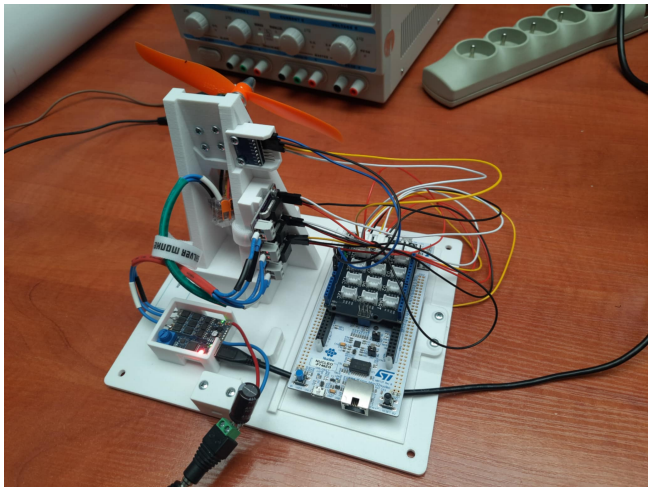

Figure 1: Rig to which this document is referring to

1 Safety guidelines

- Installed motor can reach speeds up to 19000 RPM. When turned on, the propeller may be hard to notice. Making contact with rotating propeller may cause harm to body parts or the rig itself.
- When the inverter is supplied with power, do not place your hands or any objects within the reach of installed propeller. THE MOTOR IS KNOWN TO START BY ITSELF, this happened few times when the inverter, which already had been supplied with power from external source was then connected to a laptop by a USB cable.
- When additionally weighted propeller is installed, high motor speeds may cause entire rig to vibrate with extreme amplitudes.

2 Required tools and software

2.1 Hardware

- A computer is necessary in order to power the nucleo board and the inverter from its USB ports, control the motor and gather data from sensors.
- An external DC power supply is needed to power the inverter with up to 11 volts of voltage.

2.2 Software

- https://github.com/dominik-luczak/bldc-drone-rig - this repository contains various files dedicated for this rig (commit from 10th of July 2025, SHA: 00f7b0b4767bde8d30a5d8f4af1d3b4bc1bee4cc). That includes matlab scripts referred to in this manual.
- STM32Cube Programmer - this tool is used to program nucleo or STM-B-G431B-ESC1 boards.

- STM32 MC Motor Pilot - this tool is used to control the motor and block switches in the inverter.
- MATLAB - provided scripts (serial_realtime_acceleration_and_gyroscope_xyz_vectors.m and create_metadata_json.m) used to receive, plot and save data and generate metadata files are written in this language.

3 Assembly instructions

MPU6050 and three CJMCU-758 modules with acs758lcb-050b-pff-t current sensors should be connected to NUCLEO-F746ZG board as presented on image 2:

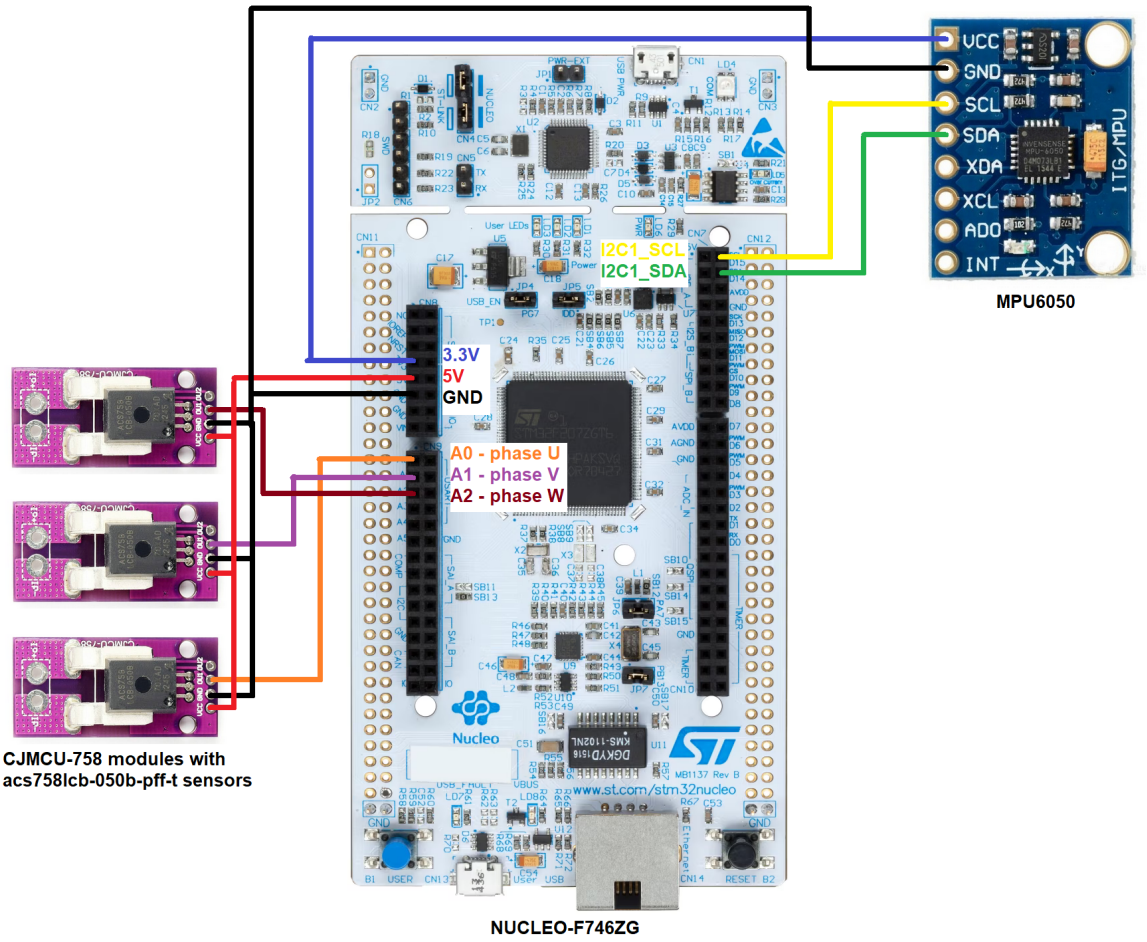

Figure 2: Sensors properly connected to nucleo board

Current sensors are vertically mounted next to each other. Those sensors have two outputs: OUT1 and OUT2. Only OUT1 is used (the one next to GND). They should be powered with 5V voltage from the nucleo board and grounded with it. Phase U is located at the bottom, furthest away from the motor. Cable with this phase is marked with white color. U phase should be connected with A0 analog input pin. Phase V lies in the middle. It should be connected to A1 pin. Phase W is the one closest to the motor. Its cable is also marked with black color. Output from this module should be connected to A2 pin.

MPU6050 module should be powered with 3.3V from nucleo board and grounded with it. SCL should be connected with PB_8/D15 nucleo pin. SDA should be connected to PB_9/D16 nucleo pin. Those pins are dedicated for I2C communication and are located in top right corner on the board.

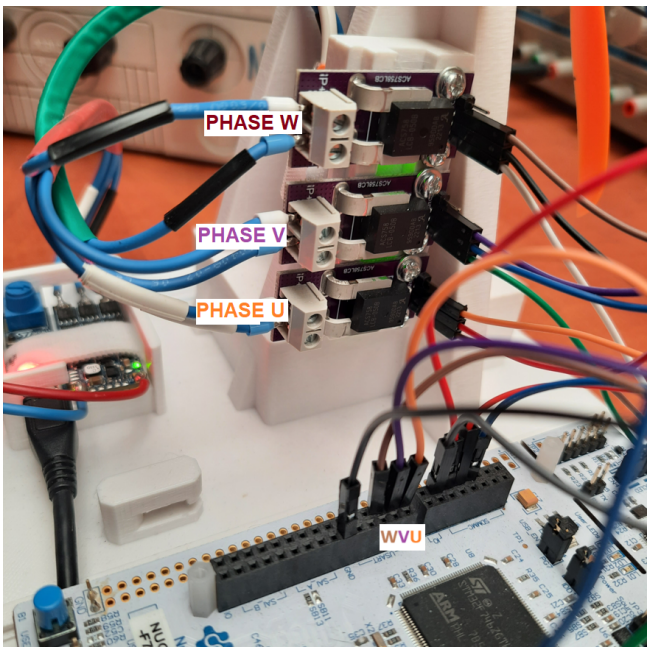

Figure 3: Labeled phases

Additionally, cables can be organized with use of shield installed on top of nucleo board. This shield (figure 4) utilizes grove connectors, which are sturdier than jumper wires thus communication via I2C is less likely to throw an error. Although this shield is causing small noise in values read by ADC.

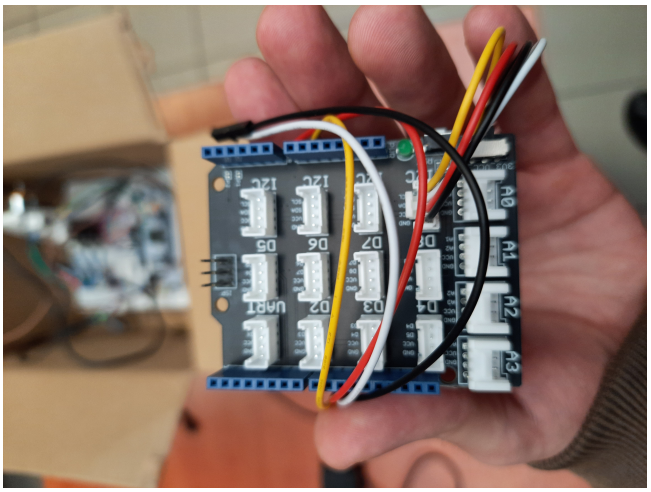

Figure 4: Shield with grove connections

Installed grove shield is labeled similarly to labels on figure 2. Note that powering voltage inside this shield's grove connectors can be set to either 3.3V or 5V via switch. Switch must be at 5 V. When the grove cables are used to power current sensors with 5V, the MPU6050 module should be powered by 3.3V with a jumper cable.

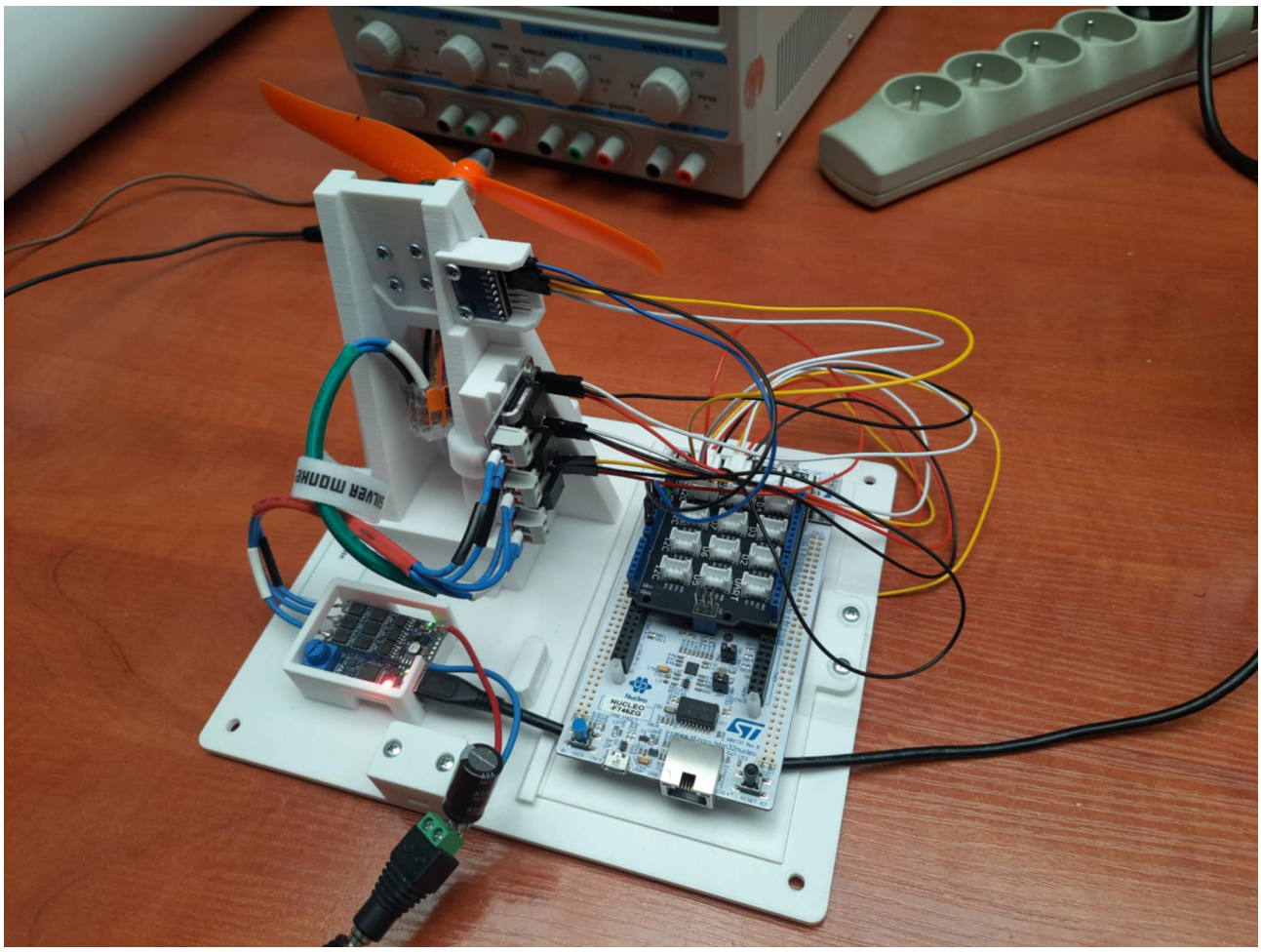

Figure 5: Sensors connected to nucleo board with grove shield

Selected propeller (normal or one of damaged ones) should be screwed onto rotor. Note that the propellers are asymmetrical and can be installed backwards.

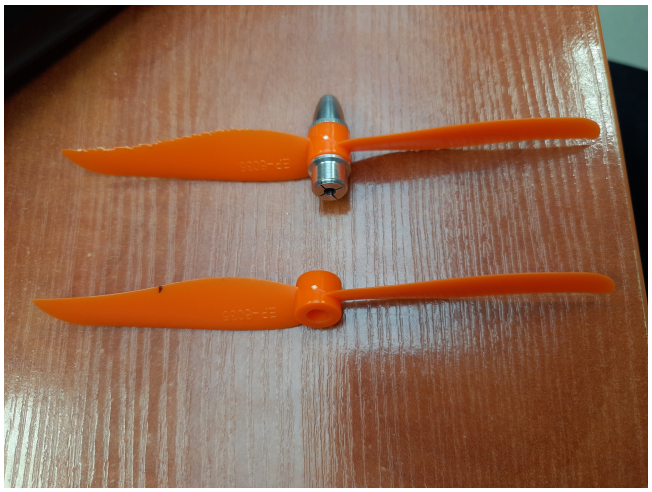

Figure 6: Normal (bottom) and damaged propeller (top)

Nucleo board and STM-B-G431B-ESC1 (inverter) should be connected to a computer with two micro USB cables. Nucleo board should be connected to a computer from its debugger's USB port.

STM-B-G431B-ESC1 board should also be powered by an external DC supply with voltage up to 11V.

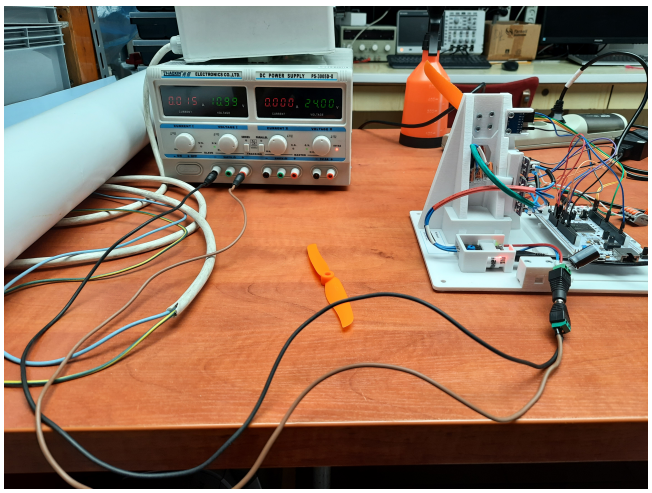

Figure 7: Supplying the inverter with 11V DC

4 Programming the boards

STM-B-G431B-ESC1 and nucleo board must be programmed to work properly. This process is performed with the STM32Cube Programmer tool.
After opening the programmer and connecting computer to chosen board with a USB cable, connection must be configured in the program. On the right side of the program, select "ST-LINK" and refresh the Serial numbers available (refresh button is shown on figure 8). After refreshing select the serial number of desired board and press "CONNECT".

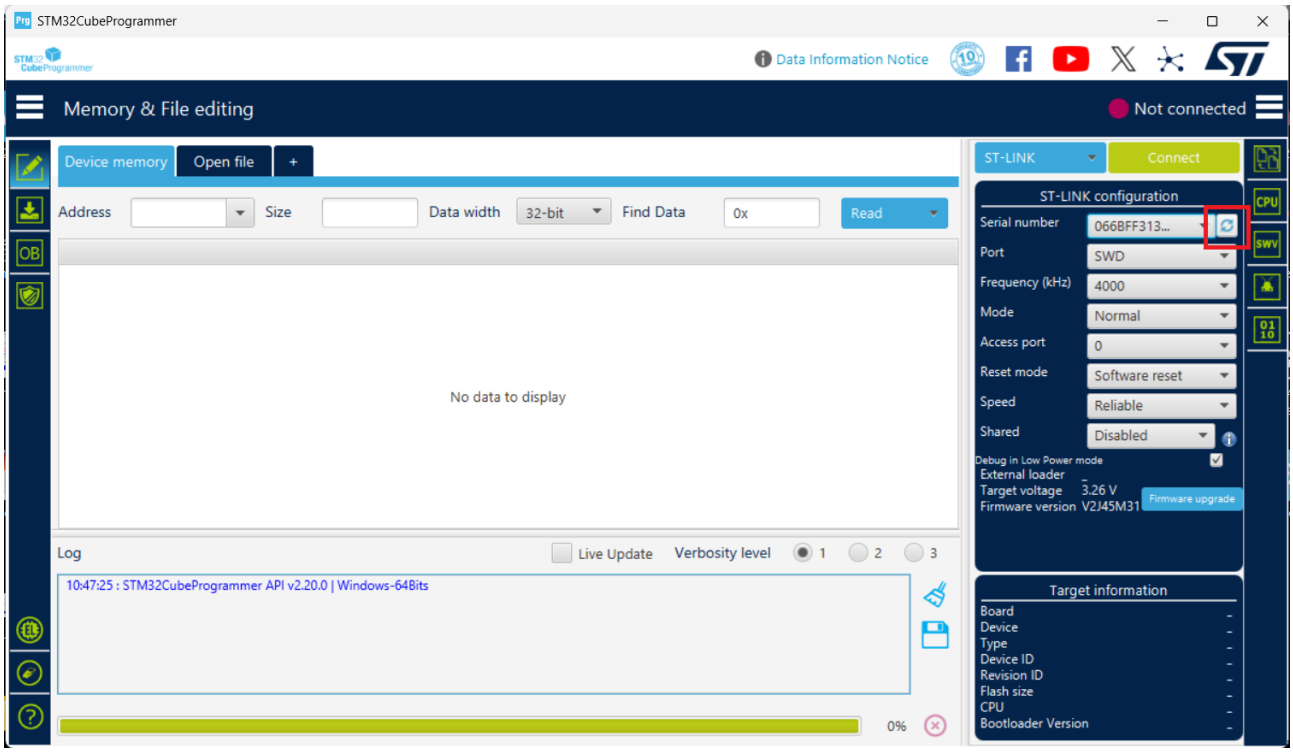

Figure 8: Connecting STM32Cube Programmer to plugged board

After establishing connection change current tab to "Erasing & Programming" tab by selecting it on the left side of the window (buttons are marked on the figure 9). In this tab, file path to a HEX file dedicated for this board must be selected. HEX files should be available in this rig's repository. After selecting the file path board can be programmed with the "Start Programming" button.

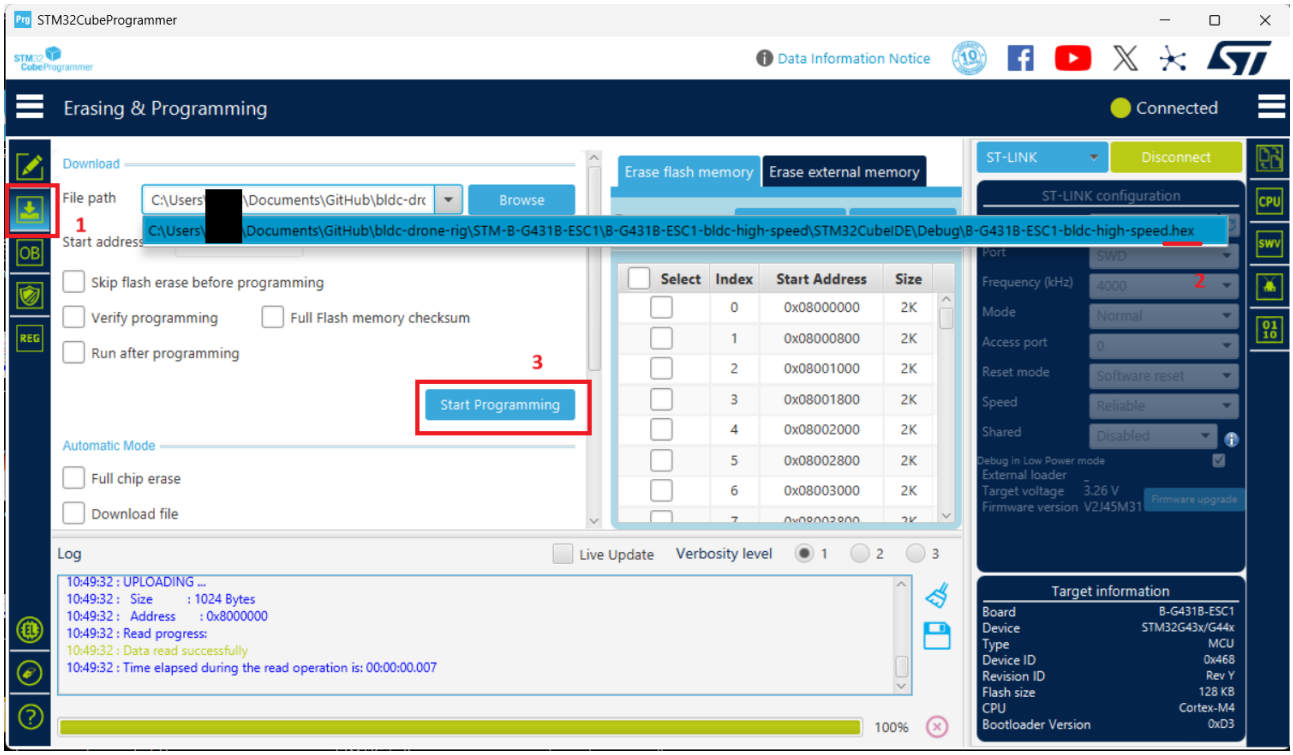

Figure 9: Programming connected board with a HEX file

5 Reading data from sensors

The MPU6050 module collects data from its 6 channels (3 axis from accelerometer and 3 axis from gyroscope) at 1kHz frequency and stores it in its dedicated internal FIFO buffer.

At the same time, nucleo board reads voltage in its 3 ADC channels at 20kHz frequency and stores it in nucleo's memory.

Nucleo board communicates with connected computer via UART. Its configuration parameters are as follows:

- Baud rate: 2250000 Bits/s,
- Word length: 8 bits (including parity),
- Parity: None,
- Stop bits: 1.

During initialization nucleo board will try to send following JSON data to connected computer:


```
{"frameSizeBytes":10568,"frameStart":[1,100,200,1],"frameEnd":
[200,30,15,200],"frameIntervalMiliseconds":80,"channels":
[{"name":"acc_x","sensor":"MPU6050","type":"int16",
"sampleCount":80,"offset":0,"samplingFrequencyHz":1000},
{"name":"acc_y","sensor":"MPU6050","type":"int16",
"sampleCount":80,"offset":160,"samplingFrequencyHz":1000},
{"name":"acc_z","sensor":"MPU6050","type":"int16",
"sampleCount":80,"offset":320,"samplingFrequencyHz":1000},
{"name":"gyro_x","sensor":"MPU6050","type":"int16",
"sampleCount":80,"offset":480,"samplingFrequencyHz":1000},
{"name":"gyro_y","sensor":"MPU6050","type":"int16",
"sampleCount":80,"offset":640,"samplingFrequencyHz":1000},
{"name":"gyro_z","sensor":"MPU6050","type":"int16",
"sampleCount":80,"offset":800,"samplingFrequencyHz":1000},
{"name":"curr_u","sensor":"ADC","type":"uint16",
"sampleCount":1600,"offset":960,"samplingFrequencyHz":20000},
{"name":"curr_v","sensor":"ADC","type":"uint16",
"sampleCount":1600,"offset":4160,"samplingFrequencyHz":20000},
```

{"name":"curr_w","sensor":"ADC","type":"uint16",
"sampleCount":1600,"offset":7360,"samplingFrequencyHz":20000}]}

This JSON contains information about data that will be transferred by UART in regular 80ms intervals. After initialization of nucleo board's program every 80ms 10568 bytes will be transferred by UART. Those 10568 bytes contain data from MPU6050 and current sensors gathered within previous 80ms.

First and last 4 bytes are constant. Those bytes are used to mark the beginning and the end of data stream. Previously mentioned JSON structure contains values of those control bytes (frameStart and frameEnd).

After 4 start bytes next 960 bytes contain values gathered by MPU6050. Each value from MPU6050 is stored in 2 bytes and should be interpreted as int16. There are 80 values from 6 channels in total ($80 \cdot 6 \cdot 2bytes = 960bytes$). The order in which values from each channel are transferred is specified in JSON structure (and shown in figure 10). Property "offset" informs after how many bytes (excluding start bytes) data from given channel will be transferred.

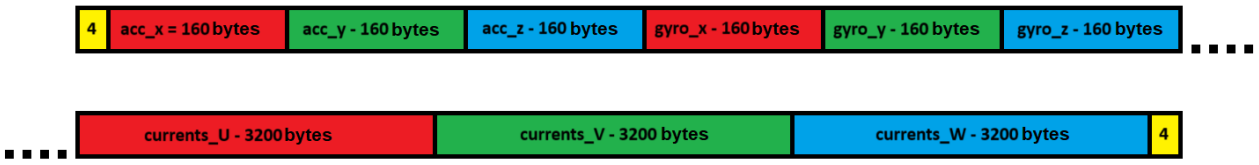

Figure 10: Structure of data transferred by UART every 80ms

After transferring data from MPU6050, next 9600 bytes contain values measured from current sensors. Currents are sampled with 20kHz frequency and are stored in 2 byte unsigned integers each (uint16). During 80ms ADC will read 1600 times each from its 3 channels ($1600 \cdot 3 \cdot 2bytes = 9600bytes$). After bytes dedicated to currents, remaining 4 constant bytes marking the end will be transferred.

Attached MATLAB script (serial_realtime_acceleration_and_gyroscope_xyz_vectors.m) can be used to plot and save this data.

This script has 6 variables declared in its first lines of code. Those variables are:

- faultType,
- setSpeed,
- dynamicStateMeasured,
- time_window_vibrations,
- time_window_currents,
- data_windows_to_save.

Variables "faultType", "setSpeed" and "dynamicStateMeasured" should be modified before every attempt at saving different data. Those variables are converted into the name of a file with data if user choses to save it.

Positive integer values should be assigned to time_window_vibrations and time_window_currents variables. They specify how much milliseconds of vibrations and currents signals will be plotted. Those variables are only for plotting received data in real time and have no effect on saving received data.

Variable data_windows_to_save specifies number of data windows (each window is 80ms long) to be saved in text file.

Port to which the nucleo board is connected must be specified by assigning a string value to the "port_name" variable (eg.: port_name='COM3') When the wrong port is selected the script will stop with an error and all currently used ports will be displayed in the console. If the computer is connected to nucleo board, one of displayed port numbers should be the correct one. On Windows operating system available all available ports can also be viewed in Device Manager (figure 11).

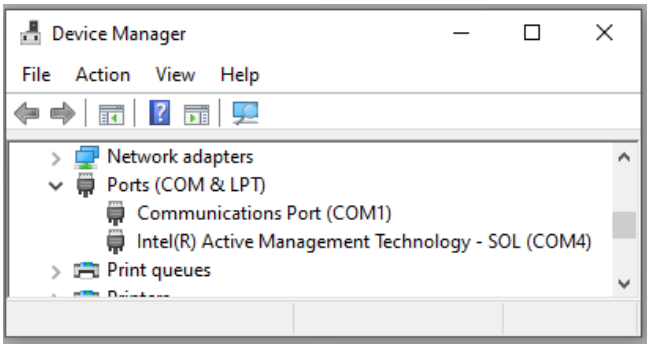

Figure 11: Checking available ports in Device Manager

In order to save received data the script must be running and the plots must be updating. Often nucleo board neads to be restarted (by black RESET button on its debugger) to start transmitting data. When the plots are refreshing saving can be started by focusing on the plot window (which can be done by selecting this window with the cursor) and pressing "S" key on computer's keyboard. The script will confirm start of saving by printing "saving started" in the console. After previously specified number of data windows will be saved, the saving will stop and message "saving stopped" will show up in the console. A text file with saved data will be created in the same folder that this script is located in. In order to save another portion of data without restarting the script, previously finished saving must be acknowledged by focusing on the plot figure and pressing "D" key. Figure 12 shows plot figure with signals received from unpowered motor as well as console with messages after saving 2 sets of data. Newly created text files are marked with a red rectangular. Have in mind that script shown in this picture despite having different name (akwizycja_danych_silnik_drona_ciagle.m) is the same script as serial_realtime_acceleration_and_gyroscope_xyz_vectors.m, just renamed. Automatically created files should not be renamed. Their names are chosen based on variables in script used to generate them.

Before or after saving whole dataset, script "create_metadata_json.m" should be used in order to create METADATA.JSON file describing the dataset. The structure defined in this script must be modified by assigning new values to it's fields based on current configuration of the rig. After running this script, created JSON file should describe state of the rig at which the data was collected.

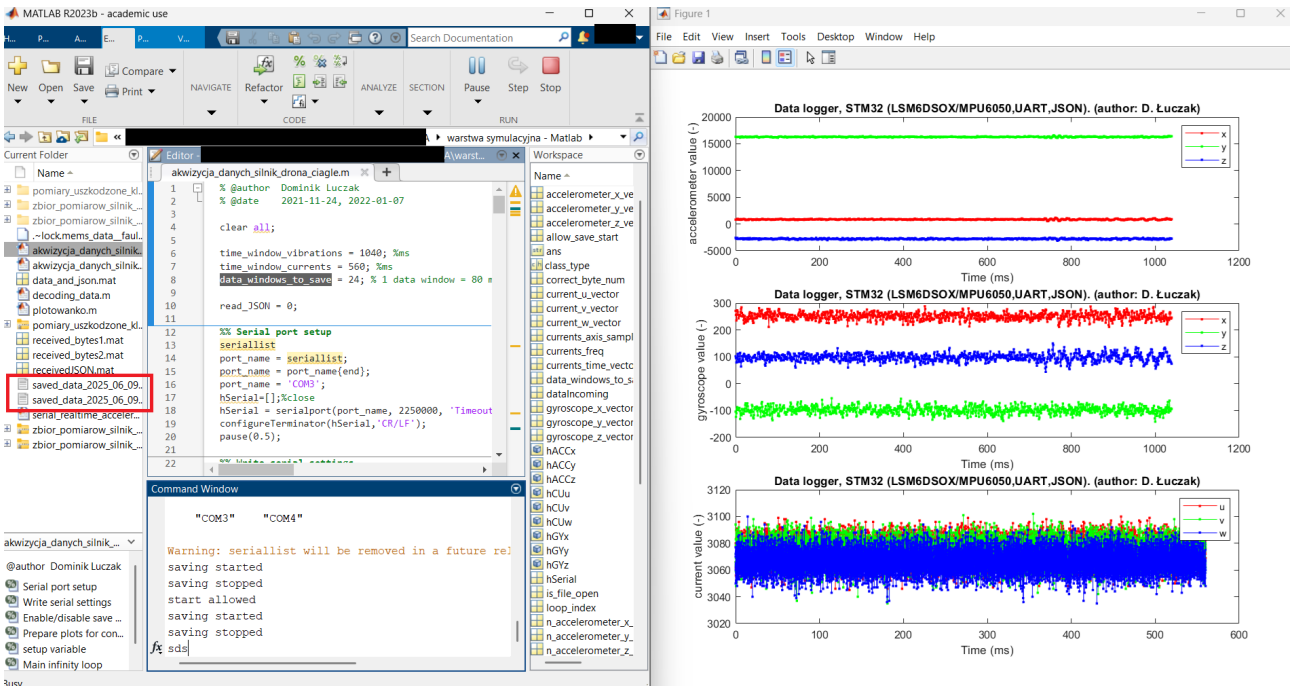

Figure 12: Saving the data

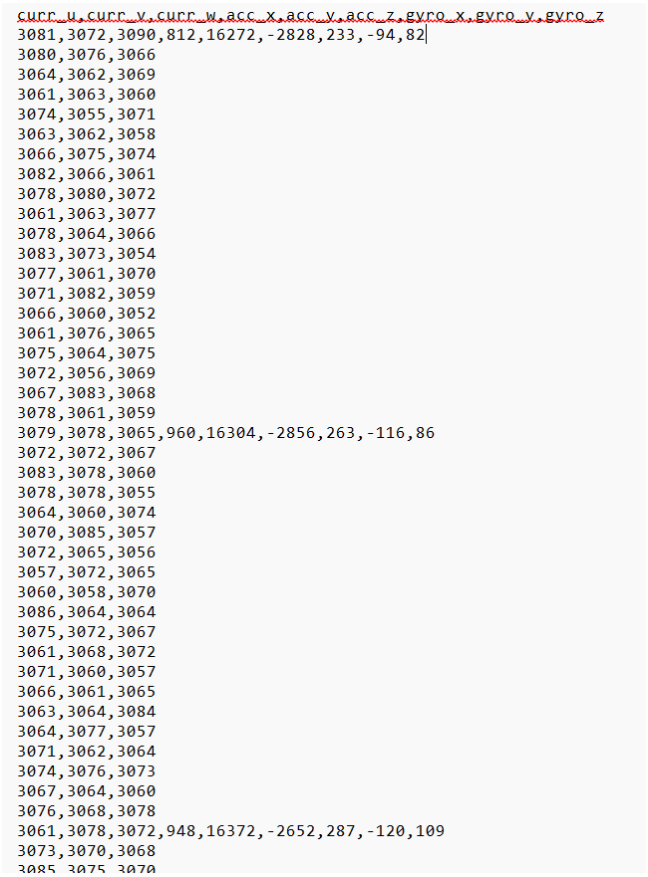

Figure 13: Fragment of a text file with saved data

6 Controlling the motor

In order to control the motor by setting its set speed and acknowledging possible errors STM32 ST Motor Pilot application is used.

After starting the application proper GUI must be selected. In upper left corner click GUI $->$ Load GUI and select "MC_FOC_SDK.qml".

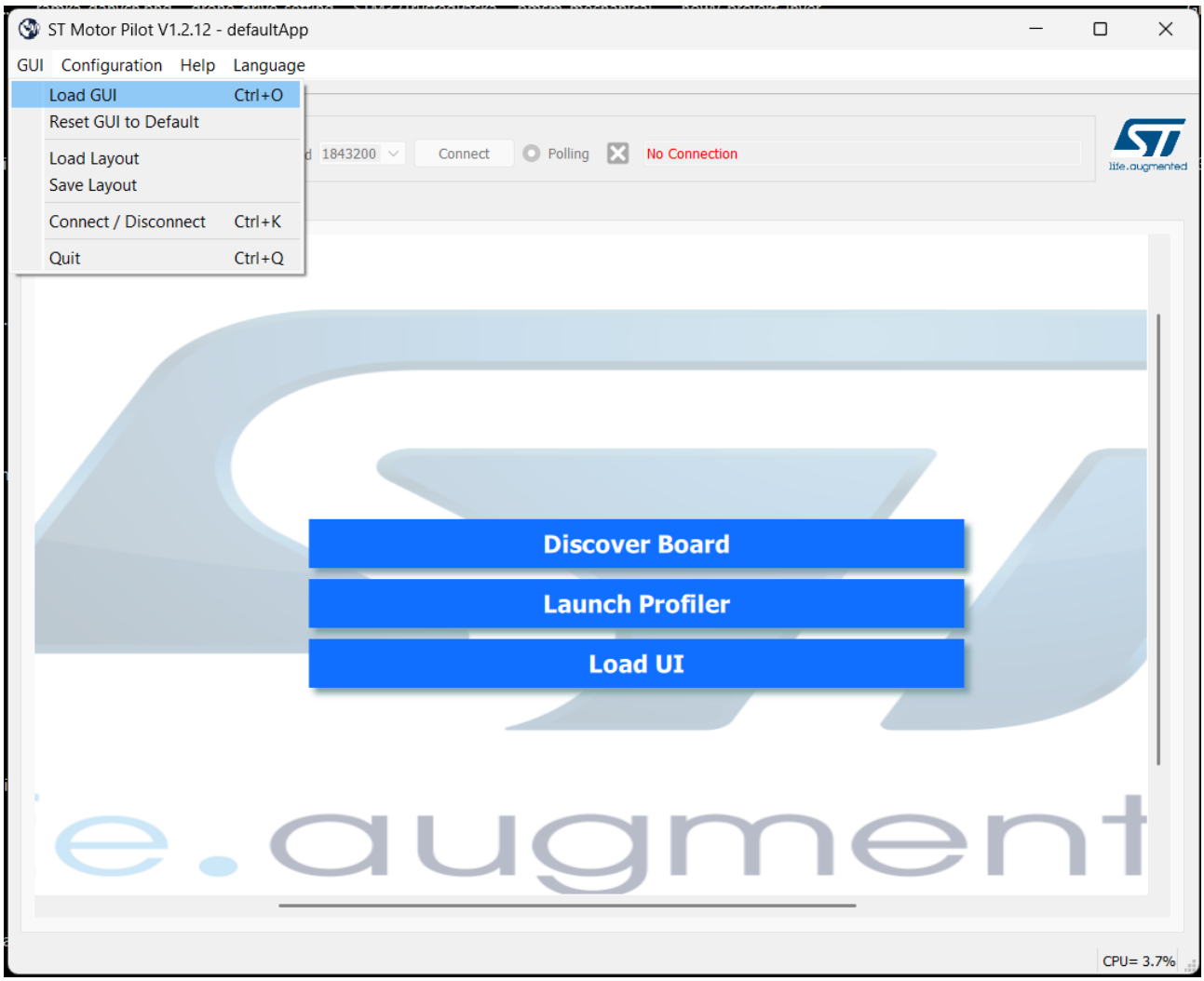

Figure 14: Loading GUI

After that, correct port must be chosen. Speed should be set to 1843200 bits per second, as shown in figure 15.

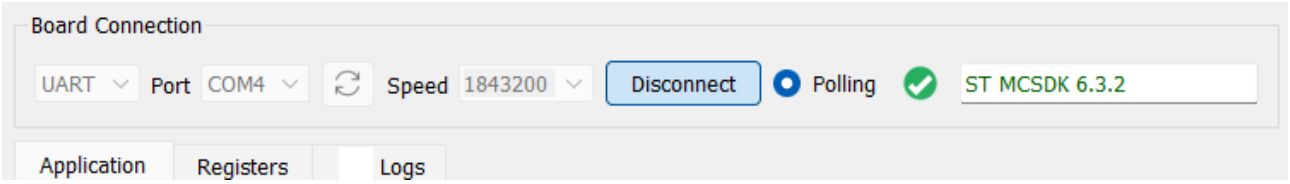

Figure 15: ST Motor Pilot Board Connection settings

After establishing connection with the inverter the motor can be turned on by selecting reference speed, acknowledging any faults and pressing the "Start" button. Speed can be set using the slider located under the speedometer or more precisely, by executing speed ramp. In order to set speed with via speed ramp input desired speed into the "Target Speed (RPM)" field and confirm it by pressing "Enter" key or by pressing "Execute Speed Ramp" Button below.

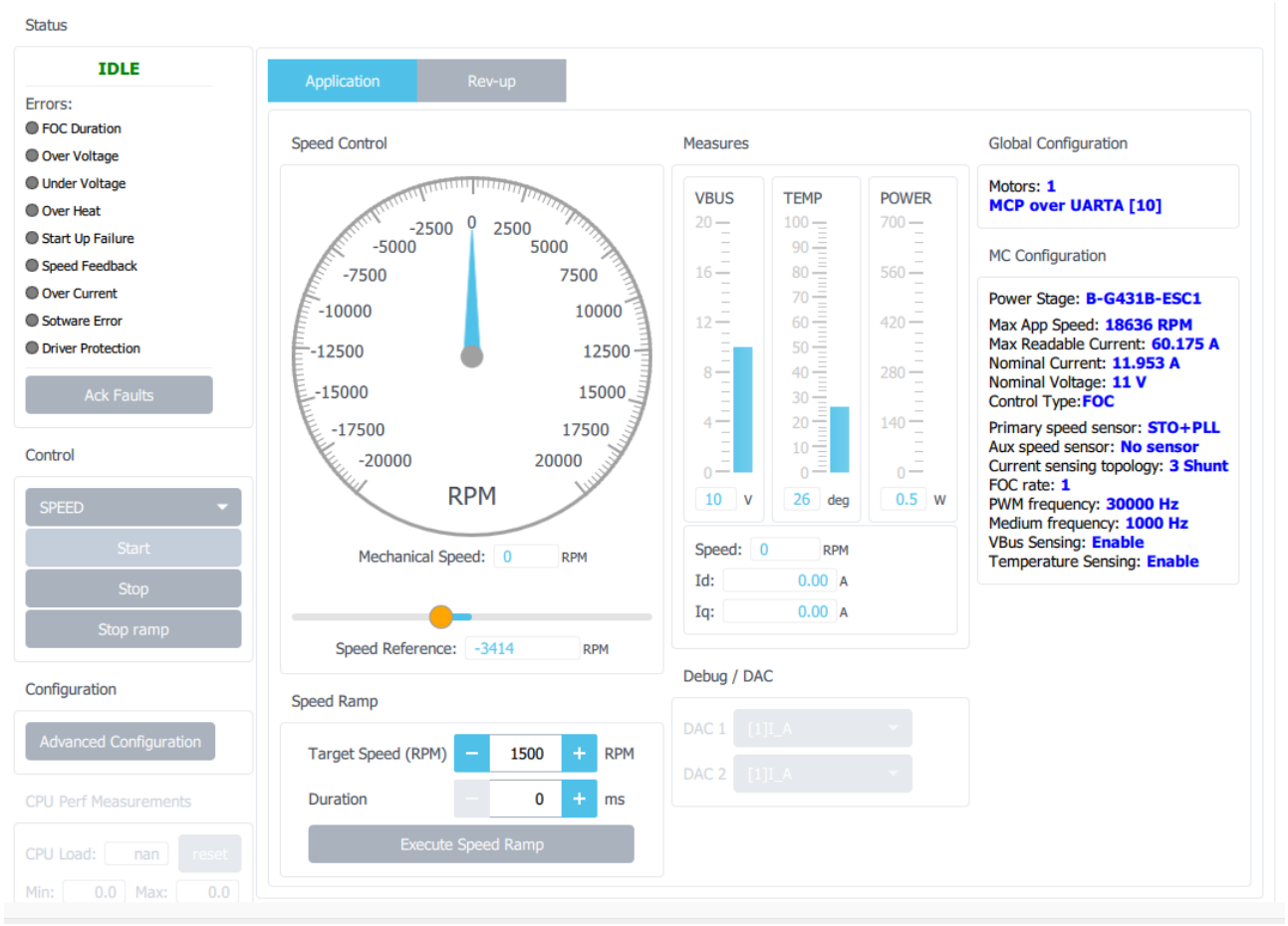

Figure 16: ST Motor Pilot Control Panel

Installed motor is a high speed motor and it might have problems running at speeds below 4000 RPM.

Starting the motor may sometimes fail and throw a "Speed Feedback" error. This happens because during the start, the motor is running in the open-loop control and only after reaching some velocity, its speed is regulated by a closed-loop speed controller. If the motor fails to start, acknowledge the error and try again. Changing starting position of the propeller may help.

7 Blocking switches in the inverter

Transistor switches in the inverter can be blocked via the ST Motor Pilot. Firstly, registers "FaultToggle" and "FaultType" must be added to the program. Register list is available under the "Registers" tab (figure 17).

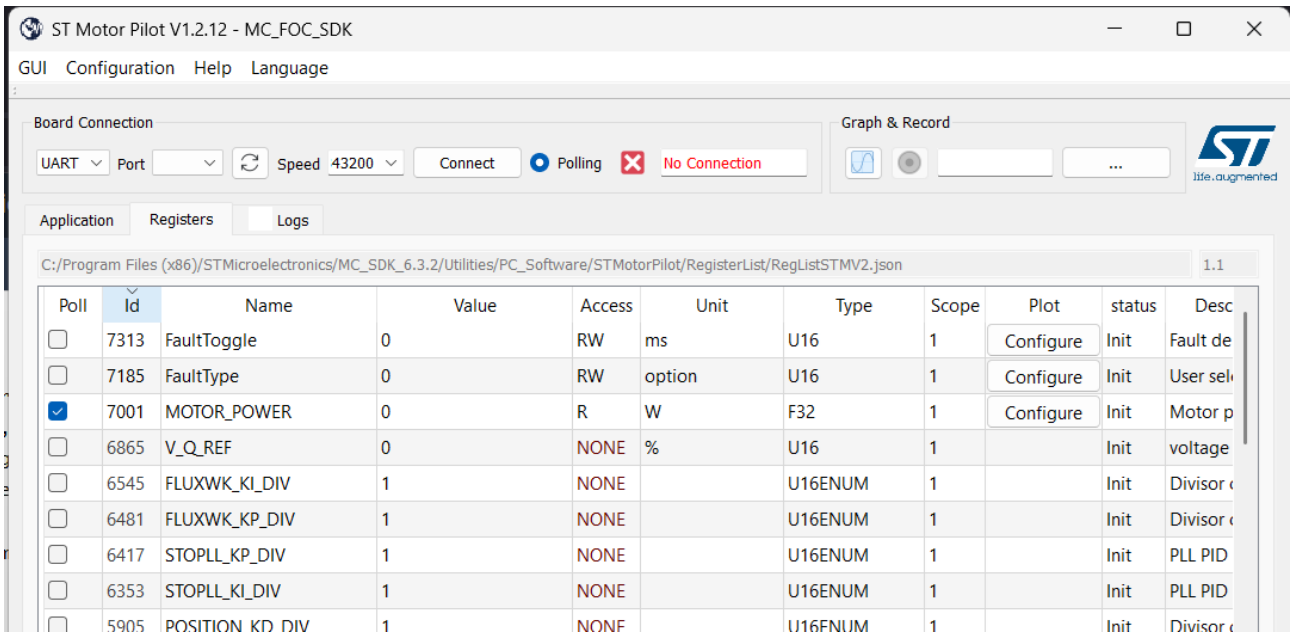

Figure 17: "Registers" tab with visible "FaultToggle" and "FaultType" registers added

Custom registers can be added by modifying "RegListSTMV2.json" file. It's path is shown above the register list. This file must be extended with two new objects shown on figure 18. This JSON file is also available in this rig's repository.

```
71  {
72    "Id": 112,
73    "Name": "FaultType",
74    "Description": "User selectable option (0-63)",
75    "Type": "U16",
76    "Unit": "option",
77    "Access": "RW",
78    "UpdateFrequency": "HighFrequency"
79  }
80  ,
81  {
82    "Id": 114,
83    "Name": "FaultToggle",
84    "Description": "Fault delay",
85    "Type": "U16",
86    "Unit": "ms",
87    "Access": "RW",
88    "UpdateFrequency": "HighFrequency"
89  }
90  ,
91  {
```

Figure 18: Two registers added to "RegListSTMV2.json"

After updating "RegListSTMV2.json" file ST Motor Pilot must be restarted. Newly added registers should now be listed with the highest ID numbers in the "Registers" tab. Sorting by Id is recommended.

"FaultType" register is used to select type of fault by modifying it's "value" field with an integer between 0 and 63. The new value can be written by pressing "Enter" key.

Inverter:

: : :

UH VH WH

: : :

UL VL WL

: : :

faultType:

0 – open circuit

1 – short circuit
fault: WL WH VL VH UL UH – binary representation from 0 to 63
ex. 35 -> 1 0 0 0 1 1 -> faults in switches WL, UL and UH
0 – no faults
63 – faults in all switches

"FaultToggle" register is used to activate the fault from the "FaultType" register. Fault activates on the rising edge after changing "FaultToggle" from 0 to 1.

In order to deactivate faults select "FaultType" as 0 and change "FaultToggle" from 0 to 1. Only one type of fault can be active at at the same time.

8 Possible issues

- nucleo board abruptly stops transferring data or doesn't transfer data at all - this problem is ofthen caused by an error in I2C communication between MPU6050 and nucleo board. It may happen during higher speeds with damaged propeller. If it happens, check the SCL and SDL connections and restart the nucleo board.
- Regular spikes of voltage are visible in three phases of currents - this may be caused by a swich-mode power supply. Try using linear power supply (the big one, visible in figure 7).
- Every 80ms, values of currents are all slightly lowered for the duration of almost whole period (about 75ms) - as for now, this is an unsolved issue. It wasn't solved because of its negligible impact on measured data. It is probably caused by a small drop of voltage used as a reference by ADC during nucleo board's computations (see VREF+ and VDDA in nucleo board manual).