ATI Mini40 DAQ F/T sensor information and tips

Iris David Du Mutel

October 9, 2023

Contents

Abstract

This document is meant to be a summary of all the relevant information found while working with the ATI Mini40 DAQ F/T sensor. It contain all the information and instructions to operate such sensor in various environments such as LabView, Python and MATLAB using the Keysight 34970A Data Acquisition Unit and the NI USB 6008 DAQ.

This document is also a description of the files contained in this project.

1 What kind of sensor is the ATI Mini40 DAQ F/T?

The transducer is a compact, durable, monolithic structure that converts force and torque into analog strain gauge signals. The force applied to the transducer flexes three symmetrically placed beams using Hooke's law (from page 16 of the manual 9620-05-DAQ):

- s = E·e
- s = Stress applied to the beam (s is proportional to force)
- E = Elasticity modulus of the beam
- e = Strain applied to the beam

Semiconductor strain gauges are attached to the beams and act as strain-sensitive resistors. The resistance of the strain gauge changes as a function of the applied strain as follows:

- ΔR = Sa·Ro·e
- ΔR = Change in resistance of strain gauge
- Sa = gauge factor of strain gauge
- Ro = Resistance of strain gauge unstrained
- e = Strain applied to strain gauge

1.1 Load calculation

From page 18 of the manual:

Additionally to this, gain correction factor is only required when a customer amplifier is being used. Refer to page 20 of the manual for more information.

2 Wiring and connecting to a DAQ

There are two different wiring alternatives for the DAQ version of this sensor:

- Differential connections to DAQ (Figure 2)
- Single-ended connections to DAQ(Figure 3)

A connection from the DAQ F/T's AGnd/AIGnd line to the data acquisition system's analog input ground or analog ground is required in most cases. This line allows the return of the small amount of current used by the data acquisition system. Noise can result if this current isn't returned via the AGnd/AIGnd path. For best noise performance, the cabling from the PS/IFPS connector should be shielded and each strain gauge's signals in a twisted pair. The shielding should be connected to the PS/IFPS connector shell and to the shell of the data acquisition system's connector. If the data acquisition system has no connector or its connector shell is electrically floating, then the shield at the PS/IPFS connector should be connected to the AGnd/AIGnd signal.

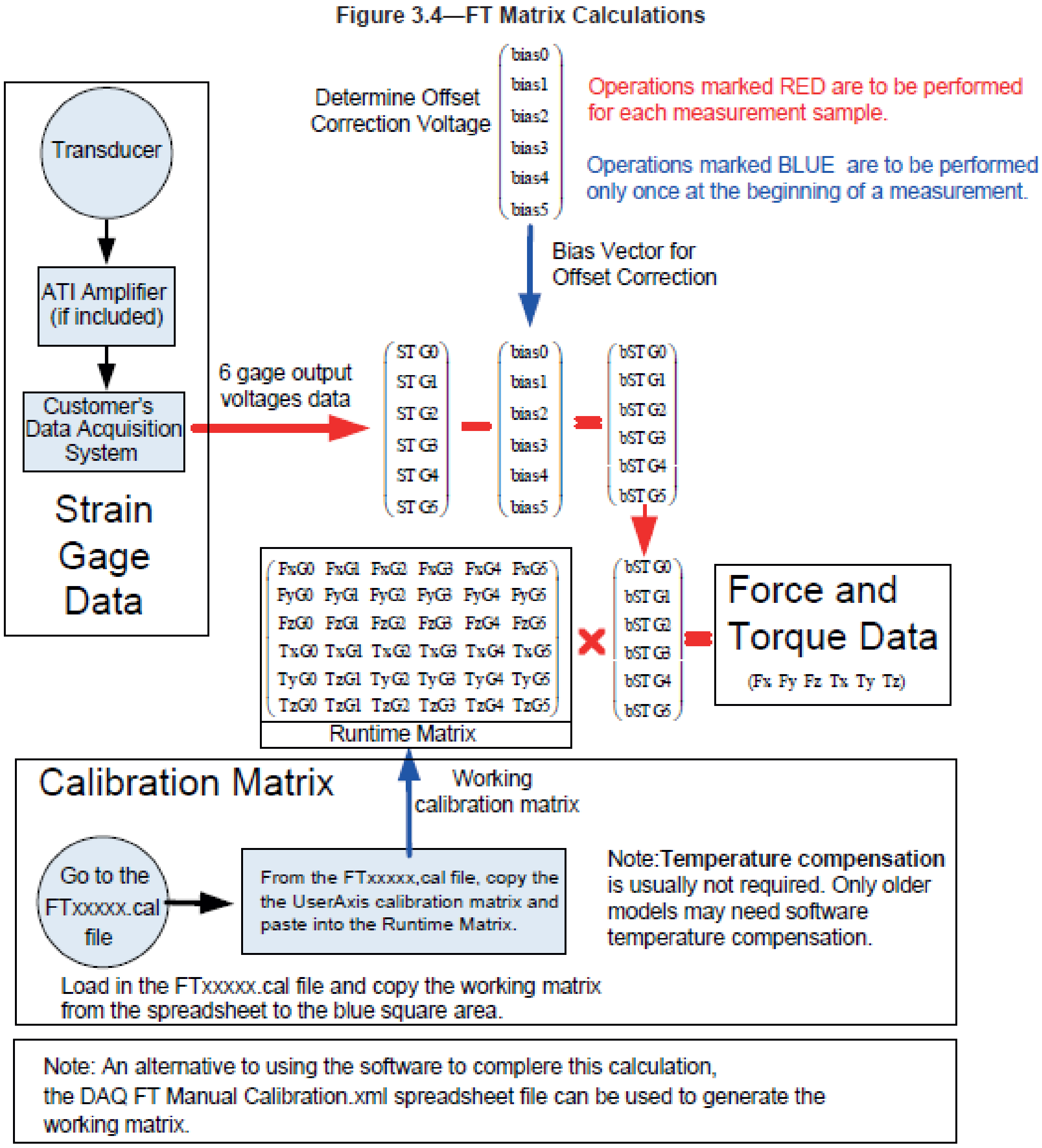


Figure 1: Load calculation process

2.1 Sampling

For best performance in all applications (page 37 from 9620-05-DAQ), the transducer electronics have bandwidth of 5kHz to 10kHz (depending on gain settings). This allows collection of all transducer frequency content. Note: that to satisfy the Nyquist Theorem, the data needs to be coupled at a rate greater than twice the highest frequency present, even if data at that frequency is not preferred. The forces and torques will be sampled at that frequency, not having anything to do with the sampling rate of the data acquisition unit.

The data acquisition unit on the other hand has a maximum aperture time of 400μs, meaning that's the smallest amount of time it needs for opening and reading from one channel. This aperture time is equivalent to an integration time of 0.02 PLC. The relationship between this two parameters is the following:

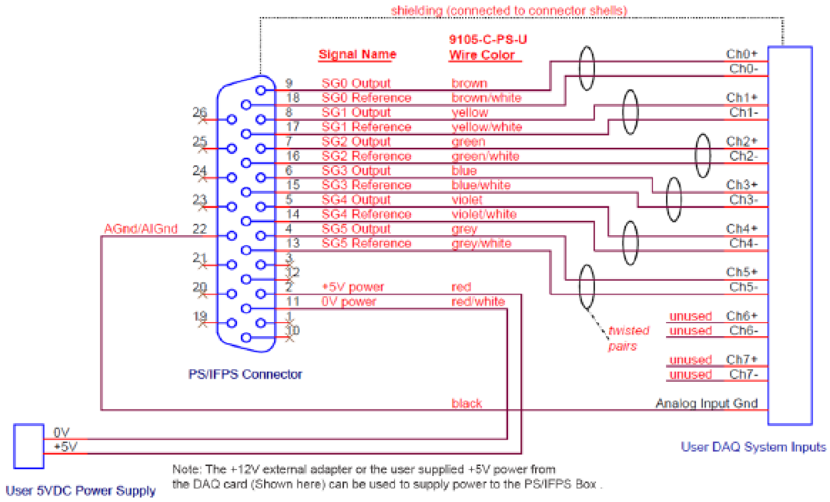


Figure 2: Differential wiring connections to data acquisition system (page 35 from 9620-05-DAQ)

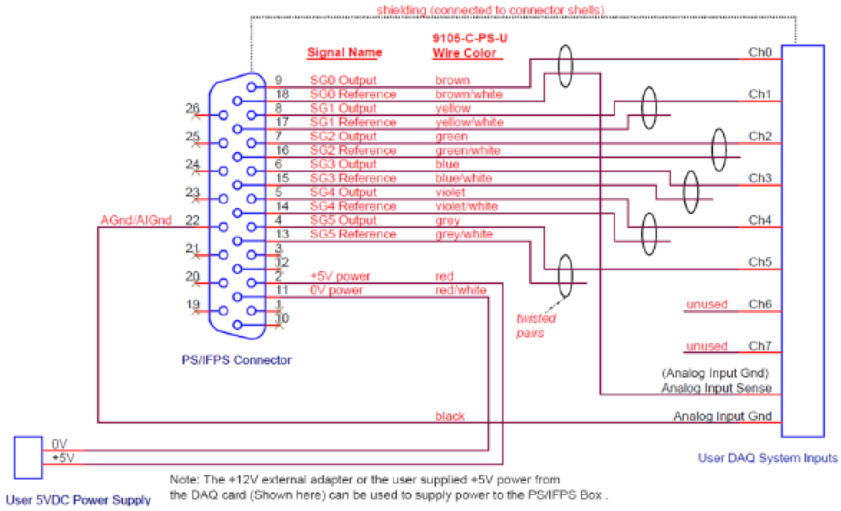


Figure 3: Single-ended wiring connections to data acquisition system (page 36 from 9620-05-DAQ)

$$\frac{0.02PLC}{50Hz(instrumentpowerfrequency)} = 400\mu s \tag{1}$$

Using a differential wiring with 12 channels, we will need to multiply that aperture time by the number of channels:

$$400\mu s \cdot 12 = 4800\mu s \tag{2}$$

An aperture time of 4.8 ms is equivalent to a frequency of around 208 Hz. Rounding the aperture time to 5 ms per data volume (a vector containing one voltage value for each channel), we can obtain 10 measurements every 50 ms.

2.2 Range

As specified in the ATI site, the range of the sensor for the calibration US-20-40 is the following defined as the average of the worst and best case scenarios:

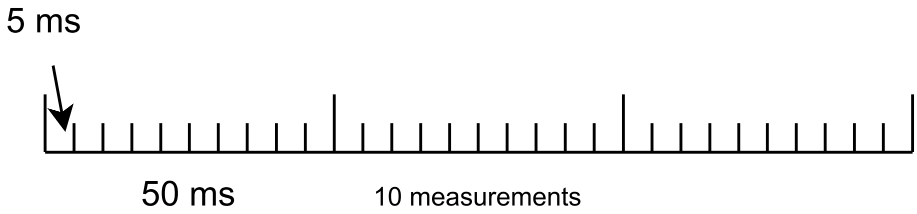


Figure 4: Representation of trigger timer of 50 ms and 10 scans of

Table 1: Range values in imperial and metric systems

Fx, Fy	Fz	Tx,Ty	Tz
± 20 lbf	± 60 lbf	± 40 lbf-in	± 40 lbf-in
± 88.9644 N	± 266.893 N	± 4.51939 N-m	± 4.51939 N-m

2.3 Resolution

As specified in the ATI site, the resolution of the sensor for the calibration US-20-40 is the following defined as the average of the worst and best case scenarios:

Table 2: Resolution values in imperial and metric systems

Fx, Fy	Fz	Tx,Ty	Tz
1/200 lbf	1/100 lbf	1/200 lbf-in	1/200 lbf-in
0.022241108 N	0.0444822 N	0.000564924 N-m	0.000564924 N-m

2.4 Sensitivity and output range and resolution voltages

From page 54 of the transducer manual, we can obtain the analog ± 10 V sensitivity. Using the data from tables 1 and 2, we can obtain the following table:

Table 3: Sensitivity, range and resolution outputs voltages (imperial system)

	Fx, Fy	Fz	Tx,Ty,Tz
Analog ± 10V sensitivity	2 lbf/V	6 lbf/V	4 lbf-in/V
Range [V]	± 10	± 10	± 10
Resolution [V]	1/400	1/600	1/800

The minimum voltage that the DAQ must be able to measure is 1/800 V = 1.25 mV. The range must be ± 10 V.

Just for clarity purposes, Table 3 in metric system would be as follows:

Table 4: Sensitivity, range and resolution outputs voltages (metric system)

	Fx, Fy	Fz	Tx,Ty,Tz
Analog ± 10V sensitivity	8.89644 N/V	26.6893 N/V	17.7929 N/V
Range [V]	± 10	± 10	± 10
Resolution [V]	1/400	1/600	1/800

3 Keysight 34970A connection to PC

The connection is made via a GPIB-USB-HS cable. The GPIB-USB-HS is an IEEE 488 controller device for computers with USB slots. The GPIB-USB-HS achieves maximum IEEE 488.2 performance. The exact model can be found in Amazon. The differences with the original true version of this device are not the sxope of this document.

There are various manuals for this DAQ. The most helpful one containing command examples is the Keysight 34970A/34972A Command Reference Manual. From this manual, the information from the following section was found.

3.1 Important commands

ROUT:SCAN : This command selects the channels to be included in the scan list. This command is used in conjunction with the CONFigure commands to set up an automated scan. To start the scan, use the INITiate or READ? command.

INStrument:DMM : This command disables or enables the internal digital multimeter. When you change the state of the internal DMM, the instrument issues a Factory Reset (*RST command).

TRIGger:SOURce : Select the trigger source to control the onset of each sweep through the scan list (a sweep is one pass through the scan list). The instrument will accept a software (bus) command, an immediate (continuous) scan trigger, an external TTL trigger pulse, an alarm-initiated action, or an internally paced timer. Usually used: TIMer = Internally paced timer trigger.

TRIGger:TIMer : This command sets the trigger-to-trigger interval (in seconds) for measurements on the channels in the present scan list. This command defines the time from the start of one trigger to the start of the next trigger, up to the specified trigger count (see TRIGger:COUNt command). A number from 0 seconds to 359,999 with 1 ms resolution. Note that 359,999 seconds is one second less than one hundred hours.

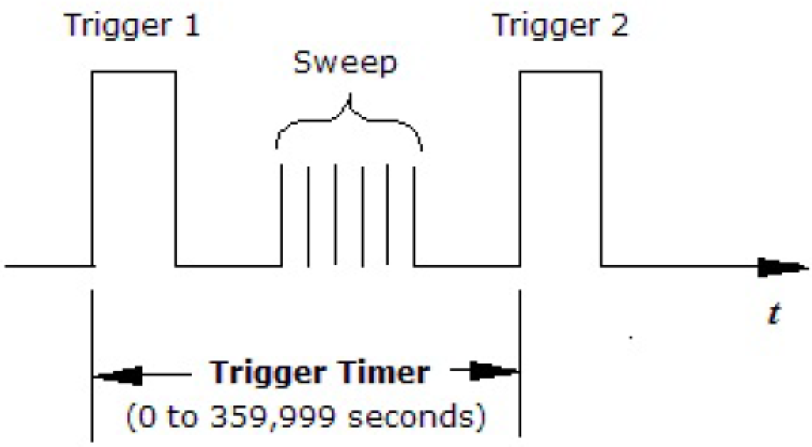


Figure 5: Trigger timer

TRIGger:COUNt: This command specifies the number of times to sweep through the scan list. A sweep is one pass through the scan list. The scan stops when the number of specified sweeps has occurred. An integer from 1 to 50,000 triggers, or continuous (INFinity).

INITiate : This command changes the state of the triggering system from the "idle" state to the "wait-for-trigger" state. Scanning will begin when the specified trigger conditions are satisfied following the receipt of the INITiate command. Readings are stored in the instrument's internal reading memory. Note that the INITiate command also clears the previous set of readings from memory. If a scan list is

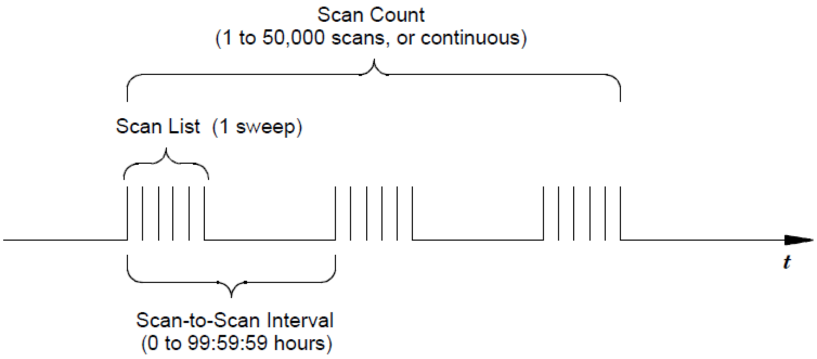


Figure 6: Trigger timer from hewlett packard manual

currently defined (see ROUTe:SCAN command), the INITiate command performs a scan of the specified channels. Storing readings in memory using the INITiate command is generally faster than sending readings to memory using the READ? command. The INITiate command is also an ”overlapped” command. This means that after executing the INITiate command, you can send other commands that do not affect the measurements. You can store up to 50,000 readings in memory and all readings are automatically time stamped. If memory overflows, the new readings will overwrite the first (oldest) readings stored; the most recent readings are always preserved. For scanning measurements using the multiplexer modules, an error is generated if the internal DMM is disabled. To retrieve the readings from memory, use the FETCh? command. The readings are not erased from memory when you read them. You can send the command multiple times to retrieve the same data in reading memory.

FETCh: This command transfers readings stored in non-volatile memory to the instrument's output buffer, where you can read them into your computer. The readings stored in memory are not erased when you read them with FETCh?.

VOLT:DC:APERTURE : This command enables the aperture mode and sets the integration time in seconds (called aperture time) for DC voltage measurements on the specified channels.

4 LabView

For LabView, a calibration file has to be loaded inside the VI. Use the file FT17838.cal and choose desired units. May not work well when using single-ended connection. In that case, use the matrices provided in ATImatrices.m under the MATLAB/Scripts folder.

4.1 Keysight 34970A

LabView offers two main ways of interacting with the Keysight 34970A DAQ:

- General purpose Virtual Instrument Software Architecture (VISA) blocks.NI-VISA is an API that provides a programming interface to control Ethernet/LXI, GPIB, serial, USB, PXI, and VXI instruments in NI application development environments like LabVIEW, LabWindows/CVI, and Measurement Studio. The API is installed through the NI-VISA driver [1].
- Agilent Technologies / Keysight Technologies 34970A drivers. These block are based on the VISA blocks but offer a more user-friendly approach to configuring the instrument as well as reading data from it.

The example provided in this repository uses generic VISA blocks. In Figure 7, the block diagram of the VI can be seen:

Inside the while loop, the write and read blocks are interacting with the instrument. Every iteration, the *write* block sends the following commands to the DAQ:

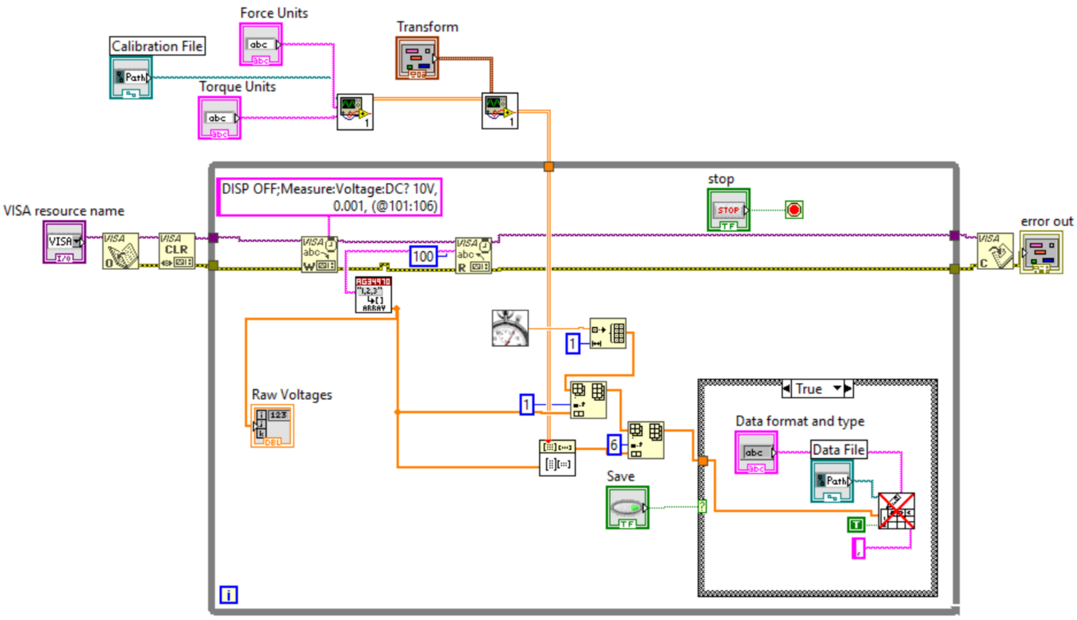


Figure 7: LabView block diagram

- DISP OFF: This command turns off the display of the external instrument. This speeds up the sampling process.
- MEASure:VOLTage:DC? 10V, 0.001, (@101:106): The first part of the command 'MEASure:VOLTage:DC?' is requesting the measurement of the voltage. The question mark indicates a query command. The two numbers following such query are the *range* and the *resolution*, respectively. There are alternative values for these parameters. See more in pages 211 to 217 from the manual.

4.2 NI USB 6008

Using the VI provided by the DAQ software download page from ATI.

5 Python

Using Qt creator, a base interface was created to interact with all python codes (see Figure 8). Since the final choice for the DAQ was the NI USB 6008, the interface has been adapted to the configuration parameters of this specific device.

There are several ways of creating a multi-channel analog reading software using Qt and the NI-DAQmx python libraries [2]:

- **NI Callback functions**:
- **QTimers** ← used in this project for ATI Mini40 and NI USB 6008 DAQ.
- **Python Threads** :

Please, visit ENGedu for fantastic code examples and references using these methods.

Temperature and pressure must be an input from the user. Otherwise, the default values will be saved.

5.1 Keysight 34790A

Using PyVisa python libraries.

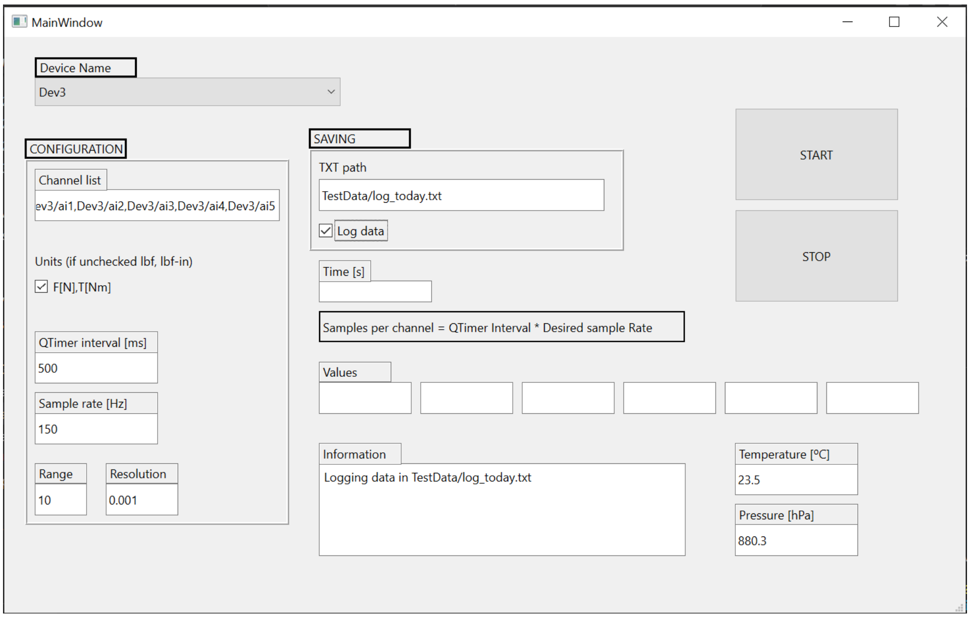
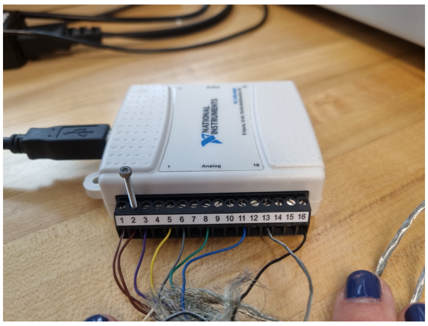


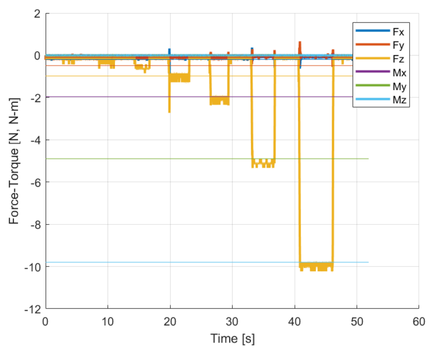
Figure 8: Qt interface for NI USB 6008 Data Acquisition Unit

5.2 NI USB 6008 DAQ

Using NI-DAQmx python libraries.



(a) Single-ended connection with only brown-white reference cable grounded



(b) Fz test check

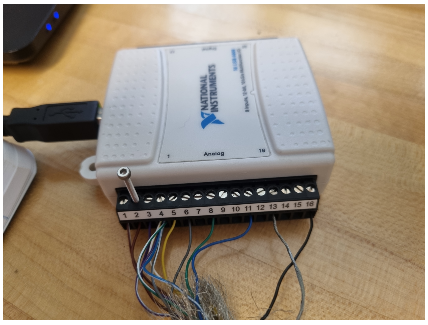
Figure 9: Single-ended connection results with non-connected references

The sampling frequency is chosen by the user. The default value is 150 Hz. In this case, being a high frequency, the Qtimer that is in charge of refreshing the GUI and in control of the python process needs to have a not so fast frequency. A QTimer of 500 ms is chosen as default. Th relationship between these two frequecies result in the following equation:

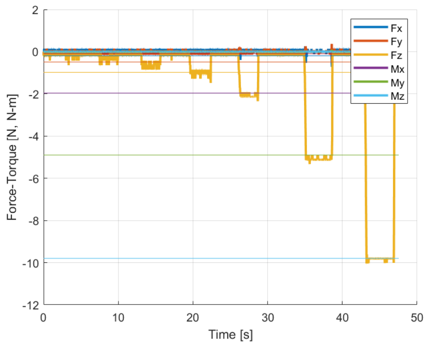
$$Samples\ per\ channel = Desired\ frequency[Hz] \cdot QTimer[s] \tag{3}$$

In the case of this application, six channels are required. Using the default values, a total of 75 samples per channel is obtained.

To log data into a *.txt* file, write the name of the desired file in the textbox and then check the *log data* checkbox (see Figure 8).



(a) Single-ended connection with all reference cables grounded



(b) Fz test check

Figure 10: Single-ended connection results with grounded references

6 SICK WLA16 Photoelectric sensors

The SICK photoelectric sensors have been chosen for this application as they have proven their applicability in UAV related research topics in the past [3]. It has to be powered with 10 V to 20 V. For more specifications regarding the power of the sensor, see the manual.

The connection type is as in Figure 11:

Figure 11: SICK WLA16 Photoelectric sensors pinout

These sensors have four cables which have the following fucntionality:

- Brown (pin 1): Positive power input.
- White (pin 2): Digital output. When and object is present between the sensor and the reflector, the signal is high. If not, signal is low.
- Blue (pin 3): GND
- Black (pin 4): Digital output. When an object is present between the sensor and the reflector, the signal is low. If not, signal is high.

The arduino connection to be used with the *sensor_analog.ino* script are as in Figure 12:

Specifications of these sensors can be found in this table:

A switching frequency of 1000 Hz implies that the sensor can complete 1000 readings in one second. In other words, it can read one complete measurement in one-thousandth of a second.

In the case of a quadcopter like the one intended to use in this work, the speed of the propellers stays in the range of 1000 up to 12000 revolutions per minute. Translating this numbers to revolutions per second, the range becomes 17 to 200 revolutions per second, which fits comfortably in the switching frequency of the sensors chosen.

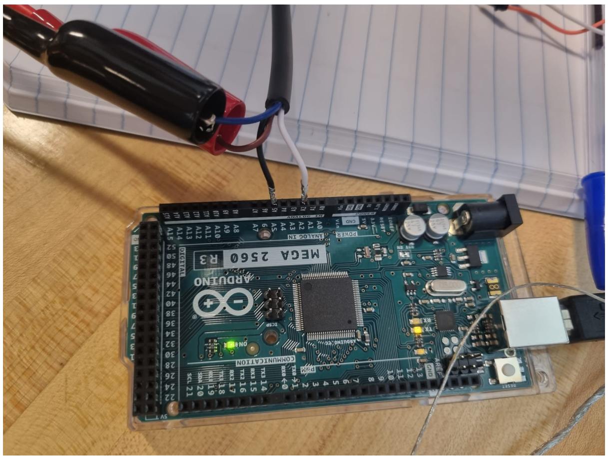


Figure 12: SICK sensor analog connection to arduino mega 2560 R3

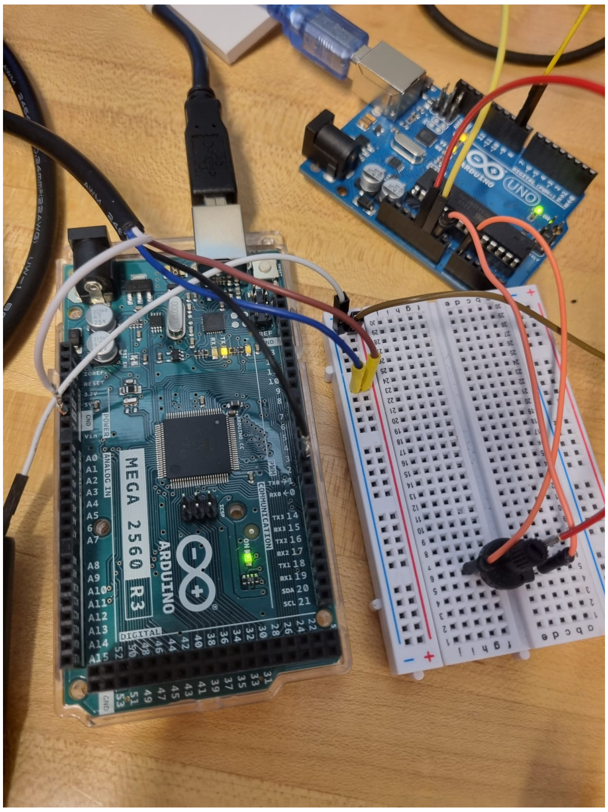


Figure 13: SICK sensor digital connection to arduino mega 2560 R3

Table 5: SICK WLA16 Photoelectric sensor specifications: digital output

Output number	2 (balck and white pins)
Signal voltage High-Low (V)	U_B-2.5V / 0 (PNP) and U_B / < 2.5 V (NPN)
Output current	$\leq$ 100 mA
Switching frequency	1000 Hz
Response time	$\leq$ 500 μs
Repeatability (response time)	150 μs

It must be said that the switching frequecy stated in the datsheet is obtained with a light/dark ratio of 1 to 1.

6.0.1 Code files

In this project, there are several arduino files that collect data from this sensor according to the connection established. Eventhoug the sensors are purely digital, the analog alternative was also tested.

The digital connection and function of 1 up to 4 sensors can be tested with the codes in the Arduino folder (tachometer_nsensor), with n the number of sensors willing to be connected.

tachometer_1sensor.ino is a script for only one sensor and uses a timer to count the time that passes in between sensor interrupts. For more information on microcontroller timers and sensor interrupts, see subsection 6.1. The same approach has been used for *tachometer_2sensor.ino*, showing that the timer system is not the best choice when dealing with more than one sensor.

tachometer_3sensor.ino and *tachometer_4sensor.ino* deal with three and four sensors, respectively by using only interrupts on three and four pins of the arduino Mega. This way, a delay is introduced for the counters to measure the sensor occurrences and once the delay is over, the elapsed time and counts are used to compute the RPM.

6.1 Timers and interrupts

Arduino Mega2650 R3 contains an ATMEGA2650 microcontroller, and a 16 MHz oscillator.

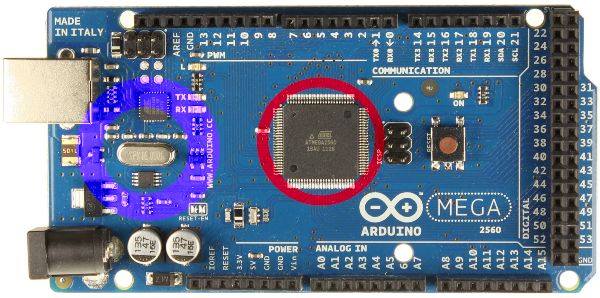


Figure 14: Arduino mega2650, the oscillator is in the blue circle and the microcontroller is in the red one

Waveform Generation mode or WGMn3:0 bits located in the Timer/Control regsiters A and B. (from page 139 of the manual) determines the counting sequence. When setting TCCR1A and TCCR1B to 0, we are choosing the *normal* mode, in which the counting direction is always incremented and no counter clear is implemented. The counter simply overruns when passes its maximum 16-bit value and then restarts from the bottom. In the case of a 16 it counter, we have the following:

$$Steps = 2^{16bits} = 65536 \tag{4}$$

The oscillator of the board can process 16000000 steps in one second, so our counter runs into an overflow every:

$$\frac{65536 steps}{16e10^6 \frac{steps}{second}} \approx 4ms \tag{5}$$

The following figure represents the counter behavior with a waveform generator in normal mode and a prescaler of 1.

A prescaler is a number used to control the overflow frequency of a counter. If the prescaler is set to 1, this means that the equation above is going to rule out counter overflow.

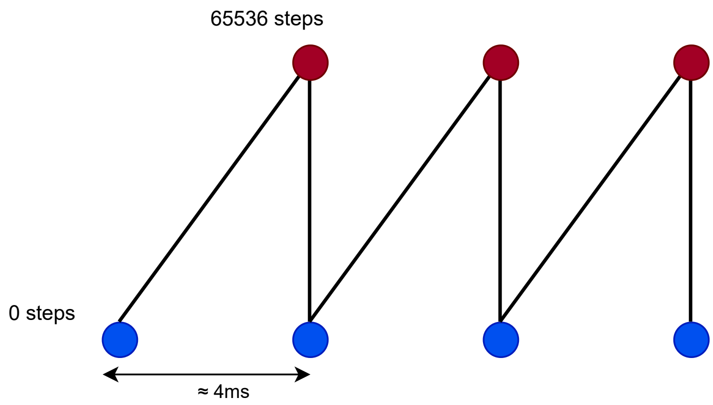


Figure 15: Counter overflow process

We can enable the counter overflow interrupt by setting the TOIE1 bit to 1. This allows us to construct an interrupt that is triggered every time the counter overflows. The prescaler of timer 1 (CS1n with n from 0 to 2) acts as follows:

$$\frac{65536[steps]}{16 \cdot 10^6/prescaler[\frac{steps}{second}]} \tag{6}$$

The table with values of the prescaler and the bits to be set is the following:

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Figure 16: Clock select bit description

This means that the minimum interrupt frequency we can obtain with a 16-bit counter is around 4 milliseconds.

There's other two ways of controlling counter overflow frequency:

- Start the counter at a different value than 0. Refer to Equation 7 to figure out how to compute the starting point
- Use clear timer on compare match. This mode can be activated by setting WGM12 to 1. It allows the user to manipulate the counter resolution by comparing the counter value to a pre-established value.

$$Start = 65536 - \frac{f_{clock}}{Prescaler \cdot f_{target}} \tag{7}$$

In the case of building a tachometer with a digital sensor, this video has been used as inspiration, changing the value of the oscillator.

In our case, using a prescaler of 256 allows us to reduce the counter maximum to 62500 (from 2^{16} to $2^{16}/256$). In this case, the time it takes for one revolution to complete can be described as:

$$\frac{counter[\frac{steps}{1rotation}]}{62500[\frac{steps}{1second}]} \tag{8}$$

To obtain the RPM value:

$$\frac{60seconds}{1minute} \cdot \frac{62500[\frac{steps}{1second}]}{counter[\frac{steps}{1rotation}]} \tag{9}$$

Depending on the value of the counter when entering the interrupt we can get a hypothetical maximum of 3750000 RPM and a minimum of 57 RPM.

6.2 Only interrupt based algorithms (FINAL CHOICE)

In the end, the use of timers to control 4 tachometers is becoming a very tedious task and readings are not correct. The final solution that was adopted is the use of four interrupts attached to the interrupt enabled pins of the arduino Mega 2650 R3 (2, 3, 18, 19, 20, 21 (pins 20 & 21 are not available to use for interrupts while they are used for I2C communication)). Increasing four counters and computing the rpm every loop iteration allows the control of the resolution and maximum and minimum values to be captured with the sensors.

The algorithm is as follows:

Algorithm 1 An algorithm with caption

Ensure: Interrupt pins: 2, 3, 18, 19 and Serial communication open

$delay \leftarrow 2000ms$
$X \leftarrow x$
$N \leftarrow n$
while $Working$ **do**
  for $t = 0$ to $t = delay$ **do**
    $counter_1 ++$
    $counter_2 ++$
    $counter_3 ++$
    $counter_4 ++$
  end for
Stop all interrupts
  $t \leftarrow Update$
  $rpm1 \leftarrow 60000\frac{ms}{1min} \times \frac{counter_1}{t_{elapsed}}$
  $rpm2 \leftarrow 60000\frac{ms}{1min} \times \frac{counter_2}{t_{elapsed}}$
  $rpm3 \leftarrow 60000\frac{ms}{1min} \times \frac{counter_3}{t_{elapsed}}$
  $rpm4 \leftarrow 60000\frac{ms}{1min} \times \frac{counter_4}{t_{elapsed}}$
  $counter_1 \leftarrow 0$
  $counter_2 \leftarrow 0$
  $counter_3 \leftarrow 0$
  $counter_4 \leftarrow 0$
end while

The resolution of the readings depends on the product of $60000\frac{ms}{1min} \times \frac{1}{t_{elapsed}}$. Depending on the delay imposed, the resolution will change. Also, the minimum and maximum values to be read will be different.

The Arduino counters have been defined as *volatile*[4] floats, which is directing the compiler to load the variable from RAM and not from a storage register. Values stored in registers can be inaccurate under certain conditions. Variables such as counters inside interrupts are being changed by something beyond the control of the code section (setup or loop code). Declaring them as *volatile*, the changes in them are immediately visible in the *loop()* function. If not, the variable state might be loaded int o a register when entering the function and would not be updated anymore until the function ends. Declaring a variable *volatile* is a directive to the compiler.

7 Pressure and temperature sensor

In this works, the barometer bmp280 has been used. It can widely measure pressure ranging from 300 mbar to 1100 hPa, with an accuracy of ±1 hPa [5]. The temperature accuracy is ±1.0°C.

In Figure 17, we can see the connection necessary to obtain the data from this sensor using the I2C wiring:

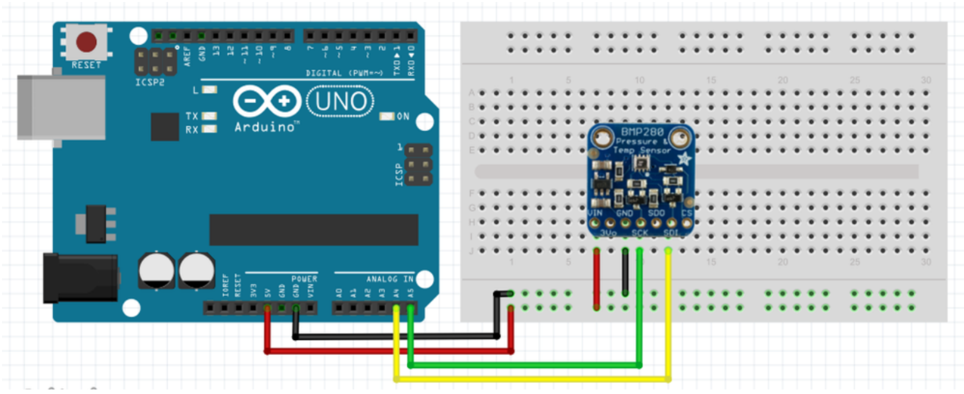


Figure 17: I2C connection between arduino UNO and bmp280 sensor

In the case of the arduino Mega, the SCK pin from the sensor must be conncted to pin SCL (pin 21) and the SDI pin from the sensor to the SDA pin from the arduino (pin 20). The SPI interface is also available for this sensor.

ADVICE: If "Could not find a valid BMP280 sensor, check wiring!" error is encountered, switch from 3.3V to 5V or viceversa in the VCC of the sensor and reload script.

8 The quadcopter

The quadcopter used in the ground/ceiling/wall effect tests is depicted in figure Figure 18 and is constituted by a metallic frame that enables different distancing between the motors. The ESC are the ones discussed in subsection 8.1, which can be powered by a 2s LiPo battery (7.4 V) up to an 8s (33.6 V). These controllers draw a maximum of 50 A.

More than one motor type is used since different propeller sizes are tested trying to stay close to a ratio of 2 between the diameter and the pitch. The proposed propellers are the following:

- 10" x 4.5" (Ratio = 2.22)
- 12" x 6" (Ratio = 2)
- 13" x 6.5" (Ratio = 2)
- 15" x 5.5" (Ratio = 2.72)

8.1 Castle Phoenix Edge 50A ESC

The manual of these ESC is in the following link.

The programming instructions were followed to obtain the following configuration:

- Settings 1: Option 4. 3.3V per cell

Figure 18: Quadcopter fram and motors with 10"x4.5" propellers

- Settings 2: Option 5. Brake Disabled (Factory Settings)
- Settings 3: Option 3: RPM Decrease
- Settings 4: Option 2: 12 kHz (Factory Settings)

Tho enter programming mode either follow the instructions on the manual or follow the next steps:

- Unplug your ESC and select full throttle with your transmitter
- Plug your ESC and wait for 5 seconds or so. You will hear a song, two beeps and the same song again. Take your throttle to medium position and wait for the song again.
- Once you've heard the song, go back to full throttle and wait for another confirmation with the same song. Then go back to half throttle.
- If at this point you should hear the song repeat itself four times. Then your have entered programming mode.
- The options will be introduced to you as a first *beep* sequence indicating the setting number and a second *beep* sequence indicating the option number. By moving the throttle to maximum position, we are accepting the option in this setting number. If accepted, the ESC will pass onto the next setting number. If rejected by taking the throttle to the minimum, the ESC will ask about the next option in the same setting number. To let us know that the ESC has understood our command, it will start emitting two beeps at a faster rate. move the throttle stick to middle position to pass onto the next setting/option query.

The ESC connection to the arduino is shown in Figure 19:

The rest of the cables are connected to a potentiometer that acts as radio controller. The **calibration** is supposed to be automatic as stated by the manufacturer in the product's site.

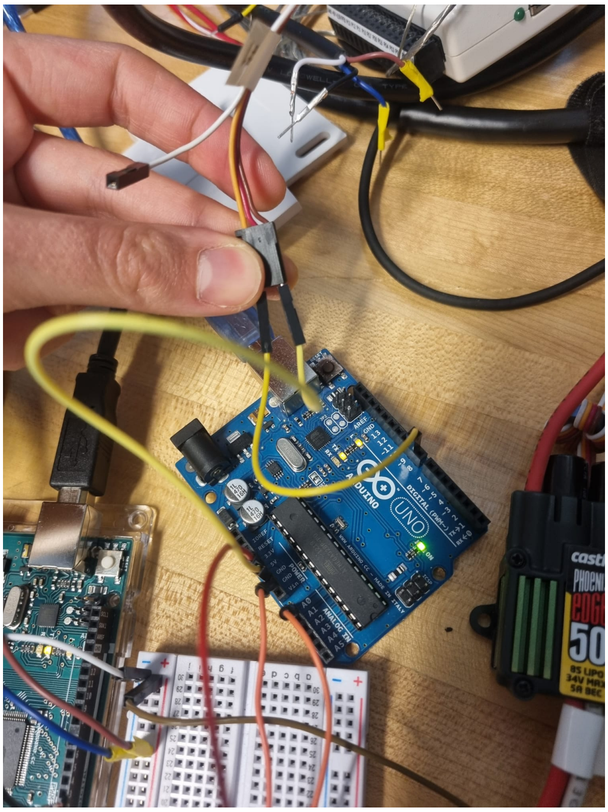


Figure 19: Brown ESC wire connects to GND and orange ESC wire is attached to pin 9 for PWM signal

8.2 Motors

The three motors chosen for testing are the following:

- Leopard 2830-12T 980Kv Brushless Airplane Motor. Used for the 10"x4.5" propellers (see Figure 20).
- Leopard 2835-12T 690kv Brushless Airplane Motor. For the intermediate sized propellers (see Figure 21).
- T-motor MN3110 470Kv brushless motors. For the bigger propellers (see Figure 22).

ALTITUDE HOBBIES

LEOPARD HOBBY LC2830 Series Specification

Model	KV	No-load current	Max Amps	Propeller	Max Voltage	Current	Pull	Power	Effciency	Load current	No.of Cells (Lipo)	Weight (motor only)**	Weight w/ hardware**	Shaft Diameter	Resistance	Length of Extended Shaft
		(A)	(A)		(V)	(A)	(g)	(W)	(g/w)	(A)		(g)	(g)	(mm)	(ohm)	(mm)
LC2830-12T	980	0.5	15	0947	7.4	5.5	419	40.7	10.3	13	2-3S	55	64	3.175	0.1575	12
					8.4	6.5	495	54.6	9.06							
					10	8.5	670	85	7.88							
					11.1	9.5	740	105.45	7.02							
					12.6	11.5	890	144.9	6.14							
					14	12.5	1010	175	5.77							
				1047	8.4	8.5	600	71.5	8.4							
					11.1	12.5	925	138.75	6.66							
					12	13.6	1000	163.2	6.13							
					14	16	1150	224	5.13							
				1147	8.4	11	720	92.5	7.78							
					11.1	16	1010	177	5.7							
					12	17	1110	204	5.44							

Figure 20: Leopard 2830-12T 980Kv Brushless Airplane Motor specifications

ALTITUDE HOBBIES

LEOPARD HOBBY

LEOPARD HOBBY LC2835 Series Specification

Model	KV	No-load current (A)	Max Amps (A)	Propeller	Max Voltage (V)	Current (A)	Pull (g)	Power (W)	Effciency (g/w)	Load current (A)	No.of Cells (Lipo)	Weight (motor only)** (g)	Weight w/ hardware** (g)	Shaft Diameter (mm)	Resistance (ohm)	Length of Extended Shaft (mm)
LC2835-12T	690	0.4	14	1147	7.4	4.8	410	35.52	11.5	12	2-4S	71	78	4	0.1758	12
					8.4	5.8	508	48.72	10.4							
					11.1	8.5	770	94.35	8.16							
					12.6	10.5	975	132.3	7.36							
					14.8	12.5	1210	185	6.54							
				1260	7.4	6.8	530	50.32	10.5							
					8.4	8	630	67.2	9.3							
					11.1	11.5	940	127.65	7.3							
					12.6	13.5	1090	170.1	6.4							
					14.8	16.5	1300	244.2	5.3							
				1340	7.4	4.8	386	35.52	10.9							
					8.4	5.5	502	46.2	10.8							
					11.1	7.5	800	83.25	9.6							
					12.6	9.5	990	119.7	8.2							
					14.8	11.5	1250	170.2	7.4							
					16	12.5	1390	200	6.9							

Figure 21: Leopard 2835-12T 690kv Brushless Airplane Motor specifications

Given the 12V and 90A limitation of the power supply, we are going to stay in a safe temperature and current range when it comes to the bigger motors (470Kv) which would require a higher voltage that would then draw lower values of current.

9 Power supply

The system needs more than one source of power to feed all boards, sensors and the motors.

The Arduinos are powered with a PC, while the sensors require a higher voltage and current.

The tachometers are powered by a Keysight triple output DC power supply (see Figure 23). A constant voltage of 10V is supplied to the sensors leaving the current bounded.

The force/torque sensor is powered by its own black box. Please, refer to page 13 from the manual for more information regarding powering this component.

The quadcopter is powered with two Astec DS550-3 power supplies connected in parallel using active load sharing. This way, a stable 12V and 90A (max) can be provided to the quadcopter, granting the repeatability of the experiments.

10 The final architecture

10.1 Data collection procedure

- Open excel file and write test characteristics
- Start logging FT data
- Start logging RPM data
- Start test script in serial terminal from arduino UNO
- Power motors
- Write temperature and pressure in excel file and wait for tect cycle to complete
- Stop all logging and make sure that everything has been saved properly

The test matrix:

Test Report			
Test Item	MN3110 KV470	Report NO.	MN.00008
Specifications			
Internal Resistance	135mΩ	Configuration	12N14P
Shaft Diameter	4mm	Motor Dimensions	Φ37.7×28.5mm
Stator Diameter	31mm	Stator Height	10mm
AWG	18#	Cable Length	600mm
Weight Including Cables	98g	Weight Excluding Cables	80g
No.of Cells(Lipo)	3-6S	Idle Current@10v	0.3A
Max Continuous Power 180S	330W	Max Continuous Current 180S	15A
Load Testing Data			
Ambient Temperature	/	Voltage	DC Power Supplier

Item No.	Voltage (V)	Prop	Throttle	Current (A)	Power (W)	Thrust (G)	RPM	Efficiency (G/W)	Operating Temperature (℃)
MN3110 KV470	14.8	T-MOTOR 13*4.4CF	50%	1.5	22.20	290	3300	13.06	40
			65%	2.6	38.48	410	4000	10.65	
			75%	3.5	51.80	550	4600	10.62	
			85%	4.9	72.52	680	5200	9.38	
			100%	5.8	85.84	780	5500	9.09	
		T-MOTOR 14*4.8CF	50%	2.0	29.60	370	3000	12.50	50
			65%	3.5	51.80	600	3800	11.58	
			75%	5.0	74.00	790	4300	10.68	
			85%	7.0	103.60	990	4800	9.56	
			100%	8.0	118.40	1090	5050	9.21	
		T-MOTOR 15*5CF	50%	2.2	32.56	460	2800	14.13	53
			65%	4.3	63.64	730	3600	11.47	
			75%	6.2	91.76	930	4100	10.14	
			85%	8.2	121.36	1100	4500	9.06	
			100%	9.5	140.60	1220	4800	8.68	
	22.2	T-MOTOR 10*3.3CF	50%	1.5	33.30	350	5500	10.51	41
			65%	2.5	55.50	500	6500	9.01	
			75%	3.5	77.70	630	7300	8.11	
			85%	4.8	106.56	780	8300	7.32	
			100%	5.3	117.66	890	8600	7.56	
		T-MOTOR 11*3.7CF	50%	1.9	42.18	430	4900	10.19	44
			65%	3.5	77.70	690	6300	8.88	
			75%	5.0	111.00	860	7100	7.75	
			85%	6.3	139.86	1030	7700	7.36	
			100%	7.4	164.28	1110	8200	6.76	

Notes:The test condition of temperature is motor surface temperature in 100% throttle while the motor run 10min.

Figure 22: T-motor MN3110 470Kv specifications

11 Post-processing

The post-processing steps are the following:

- Filtering
- Automatic detection of steps
- Offset removal

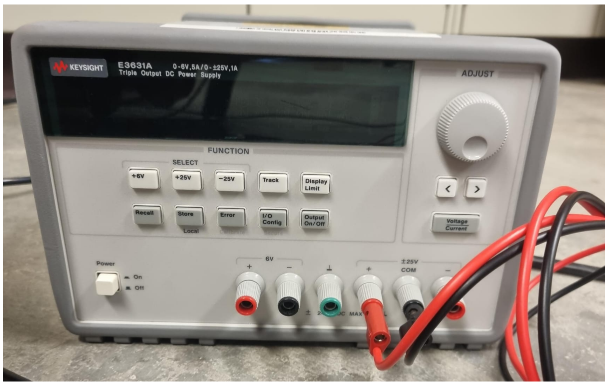


Figure 23: Keysight Triple Output DC Power Supply

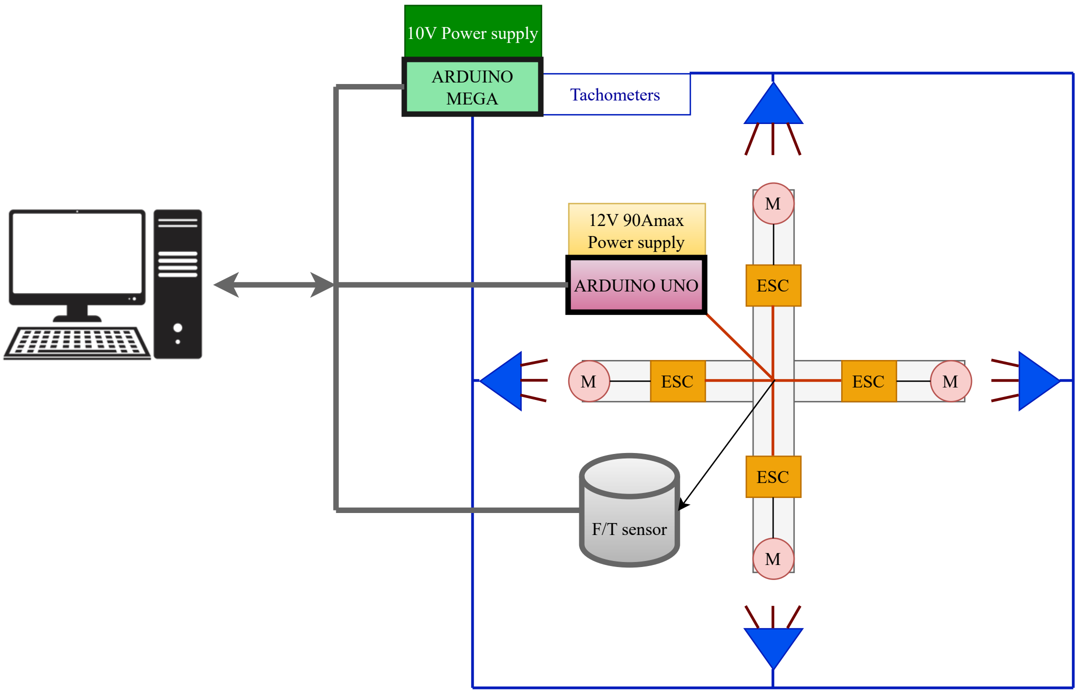


Figure 24: A graph of the overall system, representing all devices and the connections between them

Table 6: Test matrix of PWM values for each parameter and propeller size

Parameter	10”	12”	13”	15”
$\text{Fzmax}_{10”}$				
$\text{Fzmax}_{10”}/2$				
$\text{RPMmax}_{15”}$				
$\text{RPMmax}_{15”}/2$				
anything else?				

- Averaging of equivalent test data

11.1 Filtering

The data collected in this work has been obtained using the NI-USB-6008 Data Acquisition Unit and a python software and interface (go to subsection 5.2).

Notice that raw data looks like in Figure 25:

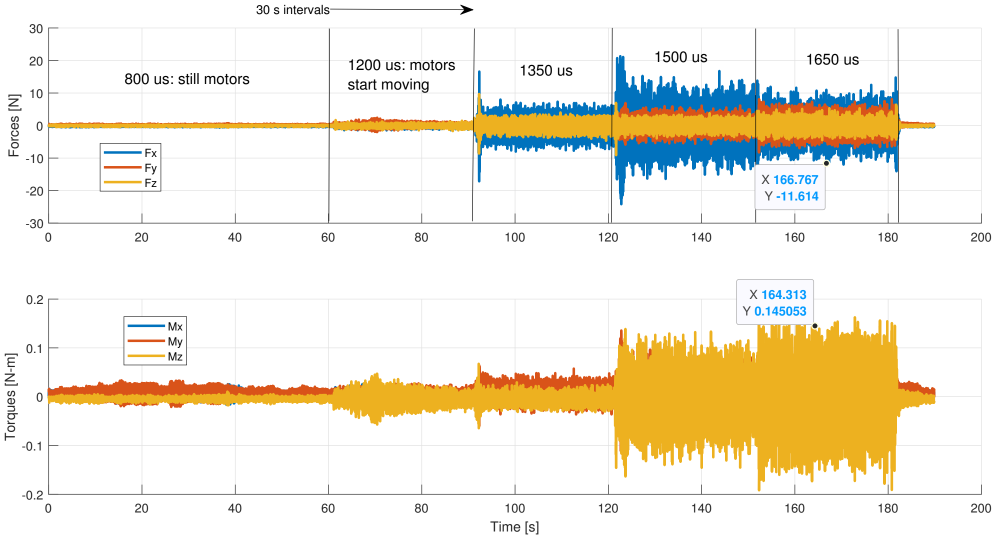


Figure 25: Unfiltered data from ascending PWM test without propellers

This forces and torques were obtained programming and ascending PWM value for 30 seconds each. No propellers were used and still noise was recorded.

For a quick example on how *Fast Fourier Transforms* and basic filtering using MATLAB functions work, see this script.

Fast Fourier Transforms help us visualize the frequencies of the data that takes part of our signals. It can be our data itself that has a charcteristic freqeuncy, or the noise and vibrations surrounding your data collection process that get captured by your sensors. In any case identifyig the desired frequencies and ignoring the rest is an absolutely necessary step i the post-processing of any data.

This is why filters are also a scope of this study[1]. The F/T sensor currently in use provides unfiltered analog data that is sampled through a DAQ, and therefore transformed into digital information. However, this digital data stream may contain noise from vibration, EM interferences... In [6], an averaging filter is implemented. The tests were all performed using a sampling rate of 1 kHz, and an averaging size of 250 samples. This resulted in an effective sampling rate of 40 Hz.

In the current work, a different approach is used. Data is sampled at 150 Hz from all 6 channels. The data is directly logged into a text file that is later or processed on MATLAB.

Using MATLAB functions such as *lowpass*, *highpass*, *bandpass* and others, one can obtain first approaches to a good filtered solution. This is a generic way of approaching the filtering problem. However, there are different types of low-pass filters:

- Butterworth filter
- Chebyshev filter
- Elliptic (or Cauer) filter
- Bessel filter

11.1.1 Butterworth filters

In Figure 26 the differences in magnitude of the different filter orders can be appreciated:

A cutoff frequency of 1 Hz was imposed and the sample frequency is supposed to be 200 Hz.

[1]For a MATLAB example on filters described in this section, look at the example script

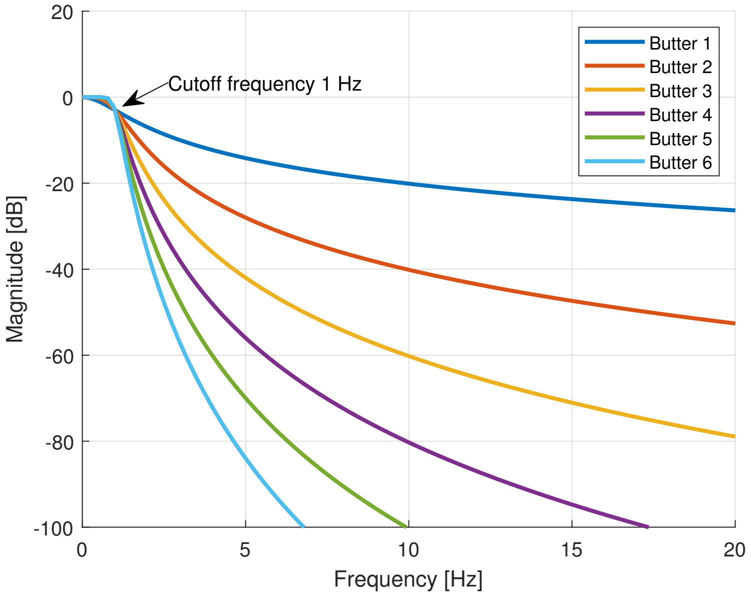


Figure 26: Butterworth filters of orders 1 to 6

The gain of the butterworth filter is as in:

$$H(j\omega) = \frac{1}{\sqrt{1 + \epsilon^2 (\frac{\omega}{\omega_p})^{2n})}} \tag{10}$$

When applied to a force vector coming from the F/T sensor, we can see the comparison between a generic low pass filter (cutoff 0.5Hz) from MATLAB and a Butterworth 5_{th} order filter in Figure 27:

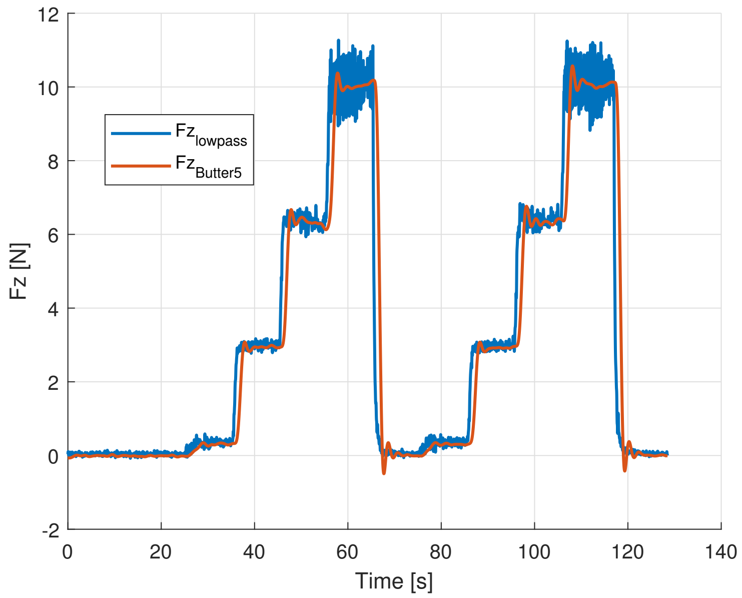


Figure 27: Butterworth filter compared to lowpass filter in MATLAB

As noticed in the picture, noise is removed successfully while a little delay is added.

11.1.2 Chebyshev filters (type I)

Using Chebyshev filters

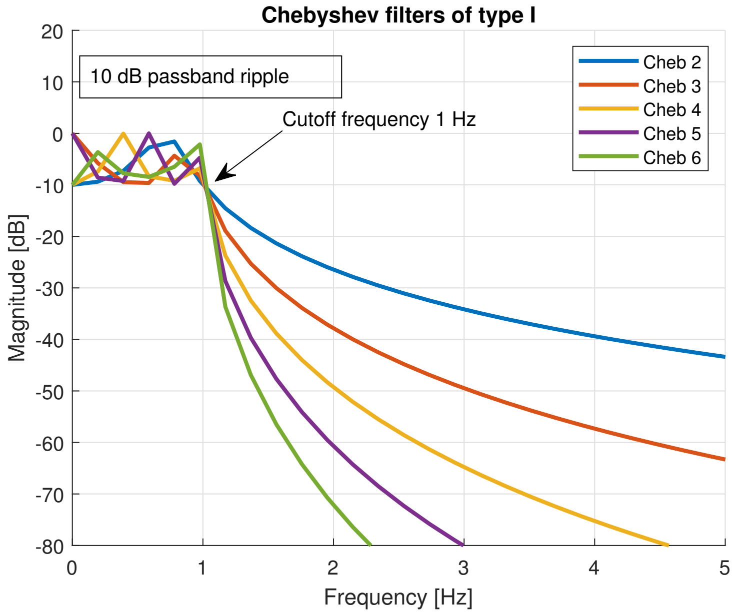


Figure 28: Chebyshev filters of orders 2 to 6

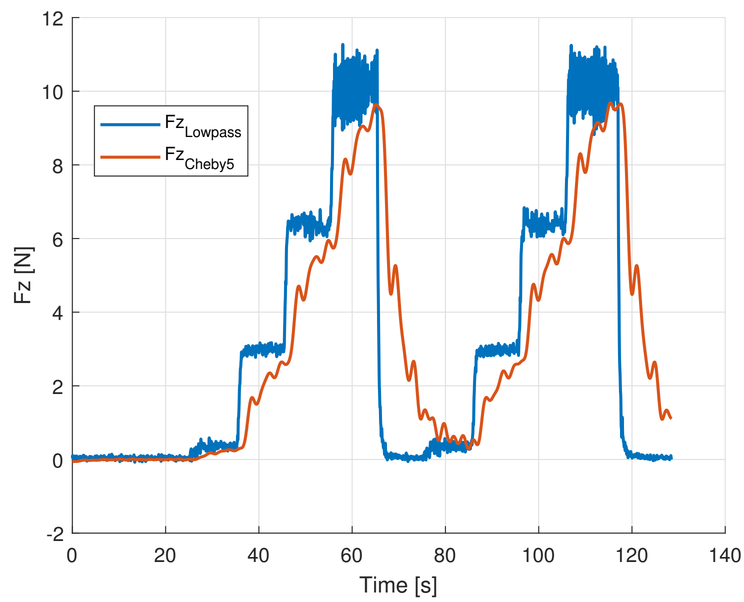


Figure 29: Chebyshev (type I) filter compared to lowpass filter in MATLAB

11.2 Automatic detection of steps

11.3 Offset removal

This part of the process is in charge of finding offsets in all datafiles and removing it.

In every test step where Fz is supposed to be 0, an offset must be removed so that the following steps are as accurate as possible.

12 Resources

Keysight 34970A/34972A Command Reference Manual

References

[1] "Ni-visa overview - ni." [Online]. Available: https://www.ni.com/en/support/documentation/supplemental/06/ni-visa-overview.html

[2] "Ni acquisition strategies – engr edu." [Online]. Available: http://engredu.com/2023/07/29/ni-acquisition-strategies/

[3] M. Scanavino, A. Avi, A. Vilardi, and G. Guglieri, "Unmanned aircraft systems performance in a climate-controlled laboratory," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 102, 5 2021.

[4] "volatile - arduino reference." [Online]. Available: https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/

[5] "Pressure sensor bmp280 — bosch sensortec." [Online]. Available: https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/bmp280/

[6] S. Conyers, "Empirical evaluation of ground, ceiling, and wall effect for small-scale rotorcraft," *Electronic Theses and Dissertations*, 1 2019. [Online]. Available: https://digitalcommons.du.edu/etd/1570