Application Report
SLUA981 – July 2019

Collecting IVT Data for Testing

ABSTRACT

This application note shows how to extract IVT (current, voltage, and temperature) data logs from a drone using the BQ40Z50-R3 battery gauge and a microcontroller for logging the data to a microSD card.

Contents

List of Figures

List of Tables

1 Introduction

To help with the simulation of a system using real-world IVT data logs, this project uses the battery characteristics of a drone in flight. The IVT logs can then be used later for simulations and other tests. The Pyboard microcontroller was chosen to extract the data from the BQ40Z50 EVM battery gauge because it can run MicroPython, it is lightweight, and is able to read data directly to a micro SD card. MicroPython has many libraries to easily read I^{2}C communication. Using I^{2}C communication, the Pyboard can log the data to a microSD card.

2 Assembling the Drone for Testing

Some modifications need to be made to the Battery Management System (BMS) circuitry of the drone to allow the BQ40z50 EVM to read the current, voltages, and temperature, along with any additional data.

2.1 Removing the Battery Casing

To remove the battery casing, locate the six plastic clips along the battery. After finding the approximate location of the clips, use a flat head screw driver to pop the hinges off. Start at the forward-most hinges and work towards the back.

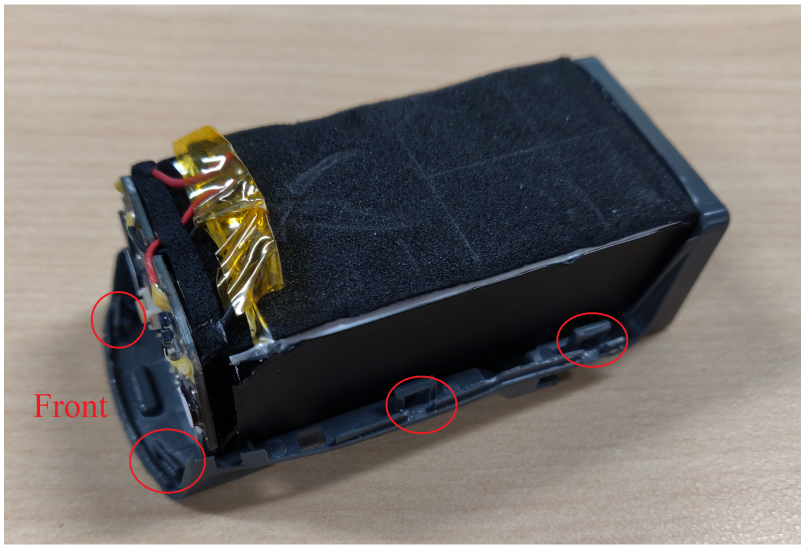


Figure 1. Battery Casing Brackets

2.2 Wire Connections

The wiring can be completed a couple different ways, depending on the BMS of the product. Figure 2 depicts the easiest way to connect an external gauge to a system. In this setup, the sense resistor for the BQ40Z50 and the existing BMS are in series. This allows both systems to run synchronously.

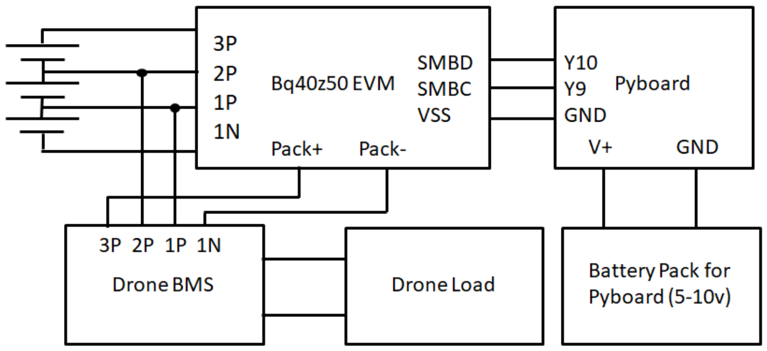


Figure 2. Overview of Wire Connections

2.3 Completed Setup

After the wiring is complete, the drone can be re-assembled for flight. A small plastic box, or other shielding, can help protect the electronics in case of a fall.

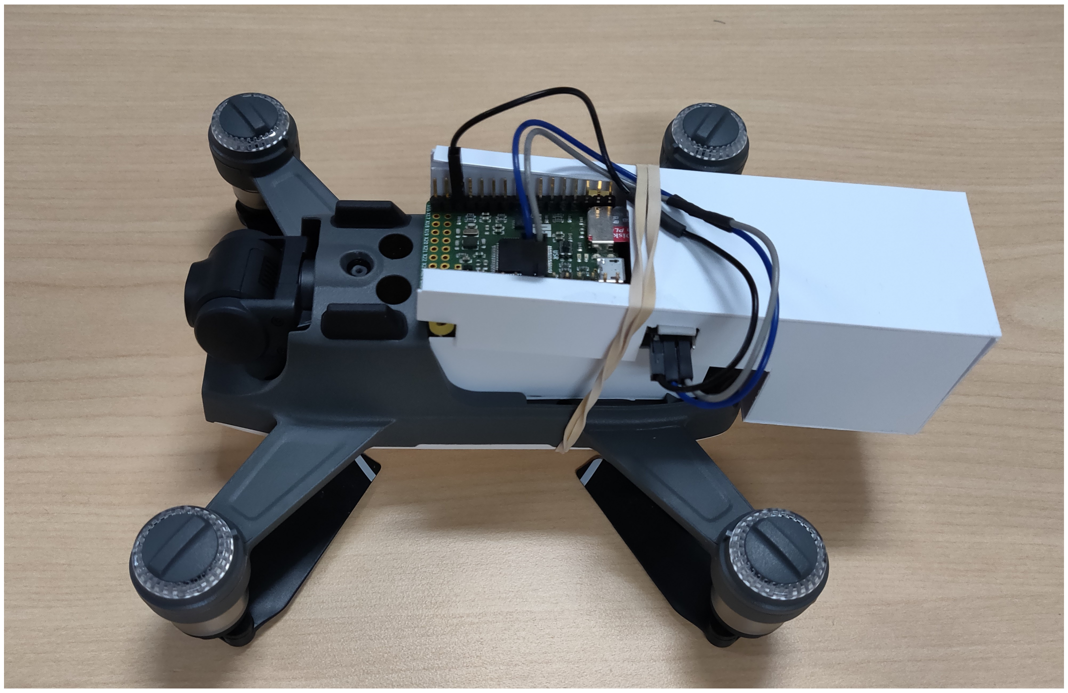


Figure 3. Completed Drone Setup

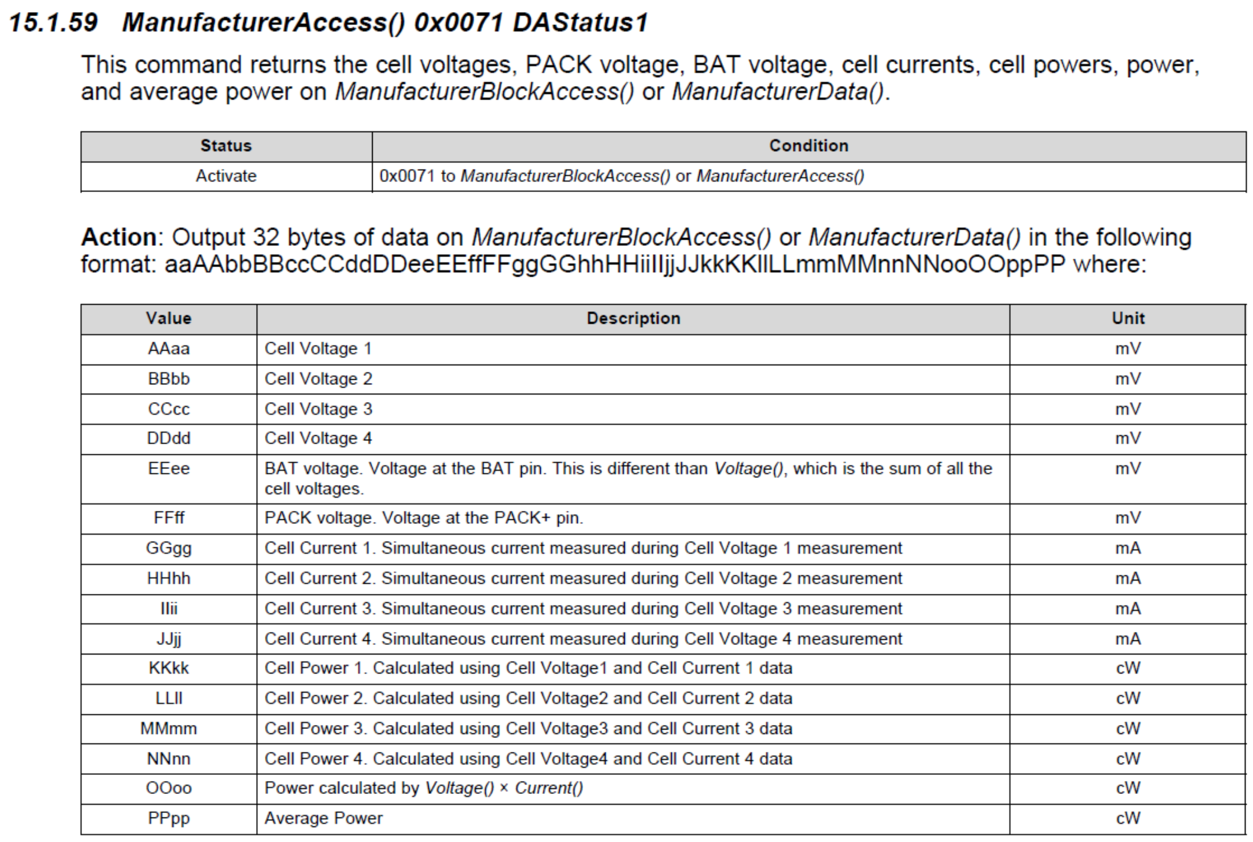
3 I^2C Interface Between the Microcontroller and the BQ40Z50 EVM

To log the IVT data and any other data, the microcontroller must use the addresses provided in the Technical Reference Manual (TRM) for the battery gauge used. Most single-cell gauges use the I^2C address of 0xAA for 8-bit addresses, while most multi-cell gauges use the address 0x16 for 8-bit addresses. In this application, the I^2C address is 0x0B because the MicroPython I2C communication library uses 7-bit addressing. The command to retrieve voltage information from data memory is 0x71 (Figure 4), 0x72 for temperature information (Figure 5), and 0x0A for the current (Figure 6).

The block commands, 0x71 and 0x72, have a size designator as the first byte, which needs to be discarded. The *ManufacturerAccess()* block commands are up to 32 bytes long to access the correct value after the byte string needs to be unpacked. To do this, a MicroPython library *struct* is used. The bytes are unpacked into a list to get the desired value the correct list designator needs to be chosen. The *"<"* signifies little endian, *"H"* is signed short, and *"h"* is unsigned short.

3.1 Gathering Voltage Data

The most important voltages for collecting data are the cell voltages and the pack voltage. The pack voltage is the voltage the load is seeing. In order to get the desired voltages out of the block command, certain bytes of data must be ignored. The first byte of the *ManufacturingAccess()* voltage block command is the size designator, which needs to be discarded.



15.1.59 ManufacturerAccess() 0x0071 DAStatus1

This command returns the cell voltages, PACK voltage, BAT voltage, cell currents, cell powers, power, and average power on *ManufacturerBlockAccess()* or *ManufacturerData()*.

Status	Condition
Activate	0x0071 to *ManufacturerBlockAccess()* or *ManufacturerAccess()*

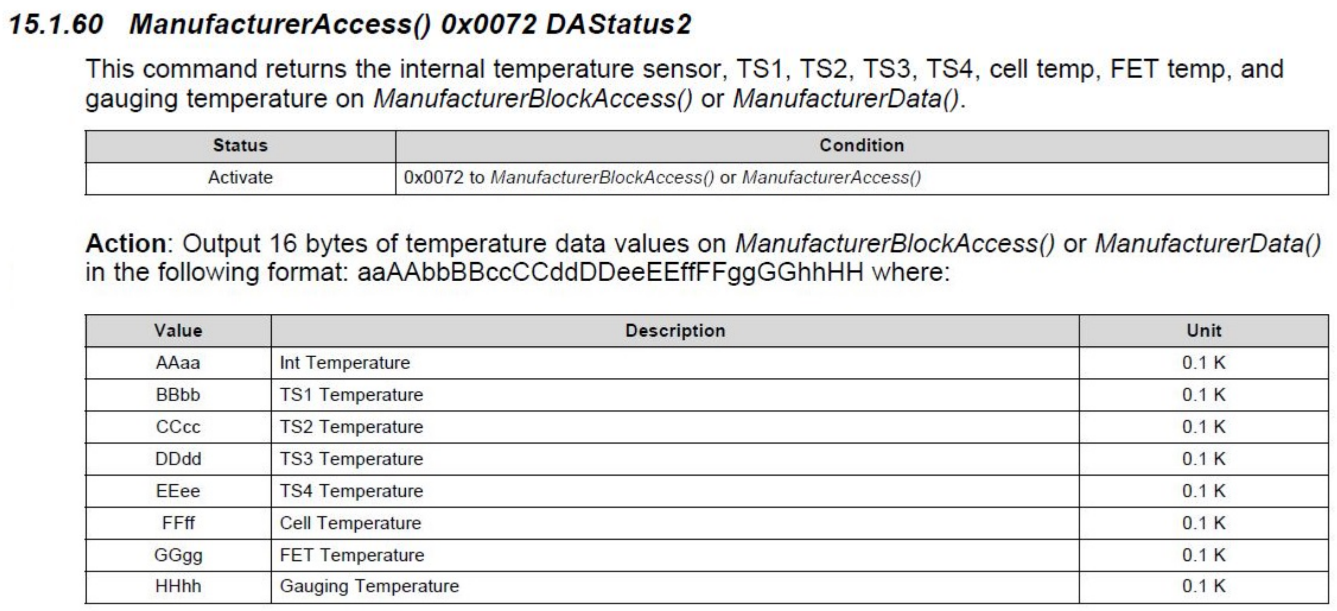
Action: Output 32 bytes of data on *ManufacturerBlockAccess()* or *ManufacturerData()* in the following format: aaAAbbBBccCCddDDeeEEffFFggGGhhHHiiIIjjJJkkKKllLLmmMMnnNNooOOppPP where:

Value	Description	Unit
AAaa	Cell Voltage 1	mV
BBbb	Cell Voltage 2	mV
CCcc	Cell Voltage 3	mV
DDdd	Cell Voltage 4	mV
EEee	BAT voltage. Voltage at the BAT pin. This is different than *Voltage()*, which is the sum of all the cell voltages.	mV
FFff	PACK voltage. Voltage at the PACK+ pin.	mV
GGgg	Cell Current 1. Simultaneous current measured during Cell Voltage 1 measurement	mA
HHhh	Cell Current 2. Simultaneous current measured during Cell Voltage 2 measurement	mA
IIii	Cell Current 3. Simultaneous current measured during Cell Voltage 3 measurement	mA
JJjj	Cell Current 4. Simultaneous current measured during Cell Voltage 4 measurement	mA
KKkk	Cell Power 1. Calculated using Cell Voltage1 and Cell Current 1 data	cW
LLll	Cell Power 2. Calculated using Cell Voltage2 and Cell Current 2 data	cW
MMmm	Cell Power 3. Calculated using Cell Voltage3 and Cell Current 3 data	cW
NNnn	Cell Power 4. Calculated using Cell Voltage4 and Cell Current 4 data	cW
OOoo	Power calculated by *Voltage()* × *Current()*	cW
PPpp	Average Power	cW

Figure 4. Voltage Byte Addresses for BQ40Z50-R3

3.2 Gathering Temperature Data

The temperature block command is very similar to the voltage command. The beginning bit is discarded and the desired thermistor temperature can be chosen in the data logging code in Figure 8. If the thermistor of interest is TS1, the list index is 2.



15.1.60 *ManufacturerAccess() 0x0072 DAStatus2*

This command returns the internal temperature sensor, TS1, TS2, TS3, TS4, cell temp, FET temp, and gauging temperature on *ManufacturerBlockAccess()* or *ManufacturerData()*.

Status	Condition
Activate	0x0072 to *ManufacturerBlockAccess()* or *ManufacturerAccess()*

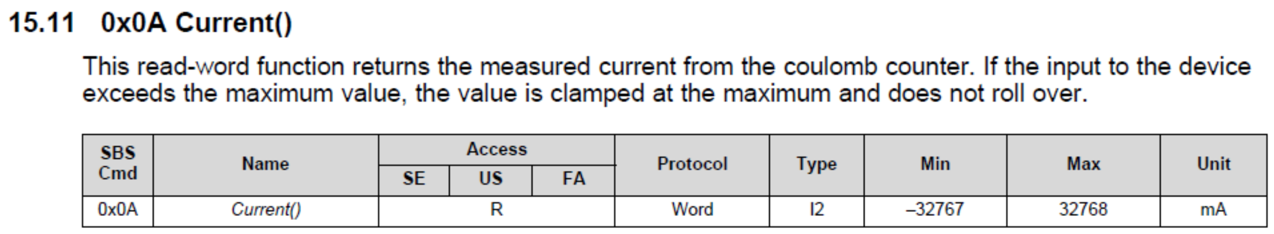
Action: Output 16 bytes of temperature data values on *ManufacturerBlockAccess()* or *ManufacturerData()* in the following format: aaAAbbBBccCCddDDeeEEffFFggGGhhHH where:

Value	Description	Unit
AAaa	Int Temperature	0.1 K
BBbb	TS1 Temperature	0.1 K
CCcc	TS2 Temperature	0.1 K
DDdd	TS3 Temperature	0.1 K
EEee	TS4 Temperature	0.1 K
FFff	Cell Temperature	0.1 K
GGgg	FET Temperature	0.1 K
HHhh	Gauging Temperature	0.1 K

Figure 5. Temperature Byte Addresses for BQ40Z50-R3

3.3 *Gathering the Current Data*

The current data is the easiest to access, as seen in Figure 8. The current is only one unsigned byte, thus it does not require the size designator byte to be discarded like the block commands.



15.11 0x0A Current()

This read-word function returns the measured current from the coulomb counter. If the input to the device exceeds the maximum value, the value is clamped at the maximum and does not roll over.

SBS Cmd	Name	Access			Protocol	Type	Min	Max	Unit
		SE	US	FA					
0x0A	*Current()*	R			Word	I2	–32767	32768	mA

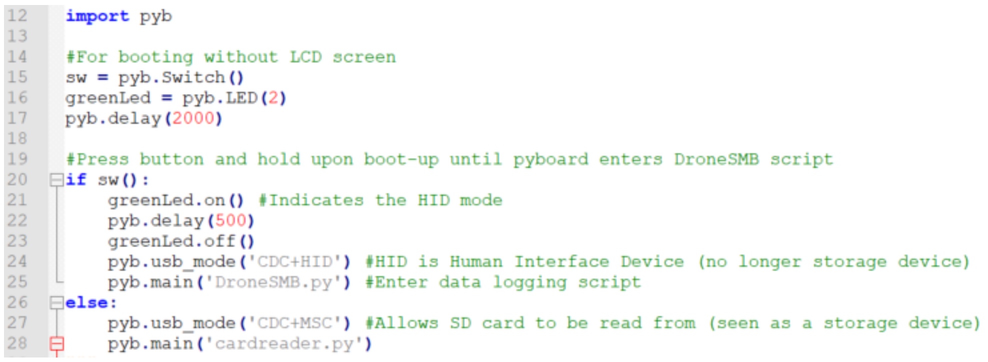
Figure 6. Current Byte Address for BQ40Z50-R3

4 MicroPython Scripts

The Pyboard uses two main scripts to collect data, the *boot.py* script and the data logging script, *DroneSMB.py*. This code can be interpreted as pseudo-code for microcontrollers other than the Pyboard.

4.1 The Boot Script

The *boot.py* script starts upon power up, or after the reset button has been pressed. In *boot.py*, the user is given the option to enter the data logging script by pressing the user button or, if no button is pressed, the Pyboard enters the script to read the SD card. If the user button is pressed, the Pyboard enters data logging mode.

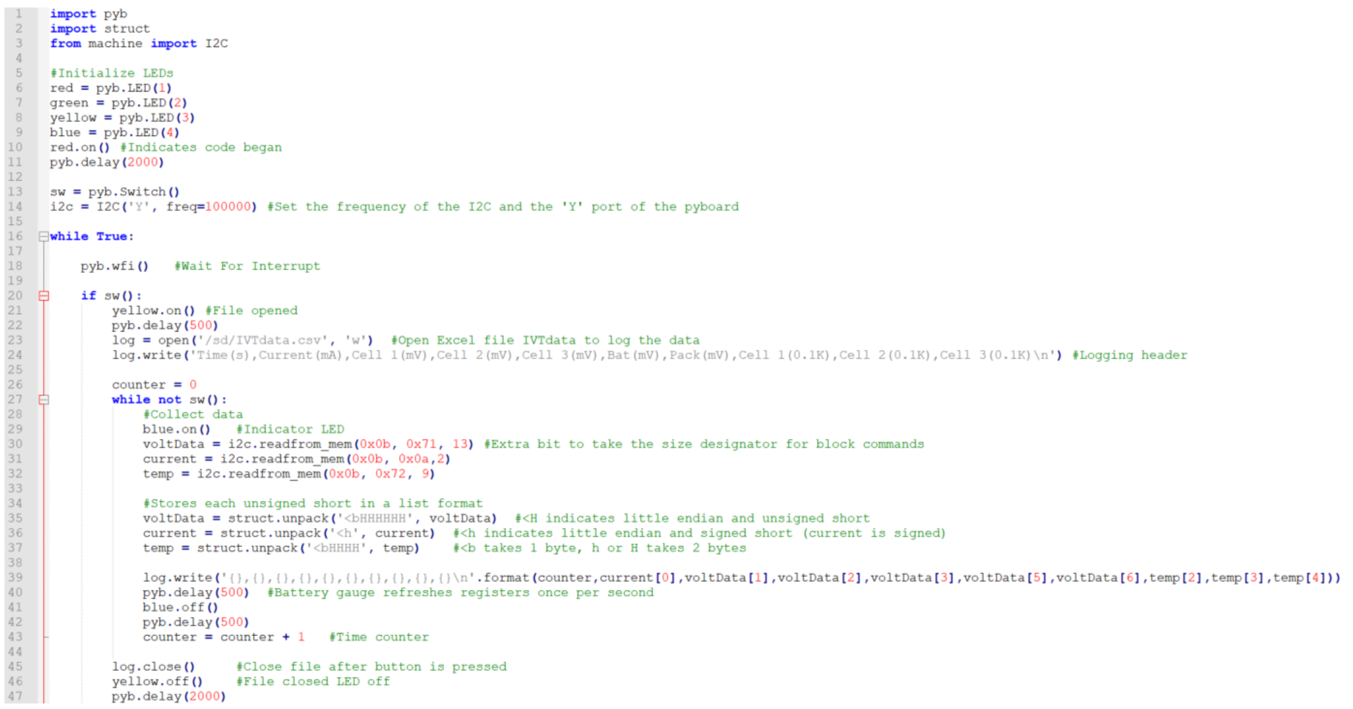


```
import pyb

#For booting without LCD screen
sw = pyb.Switch()
greenLed = pyb.LED(2)
pyb.delay(2000)

#Press button and hold upon boot-up until pyboard enters DroneSMB script
if sw():
    greenLed.on() #Indicates the HID mode
    pyb.delay(500)
    greenLed.off()
    pyb.usb_mode('CDC+HID') #HID is Human Interface Device (no longer storage device)
    pyb.main('DroneSMB.py') #Enter data logging script
else:
    pyb.usb_mode('CDC+MSC') #Allows SD card to be read from (seen as a storage device)
    pyb.main('cardreader.py')
```

Figure 7. Boot Script

4.2 The Data Logging Script

The data logging script opens a .csv file on the SD card and begins logging data. The script uses a combination of the onboard LEDs to indicate what section of the script the microcontroller is running and the push button for changing to the next part of the script. Data is logged every second because the data registers are updated every second. Anything faster can lead to redundant data.



```
import pyb
import struct
from machine import I2C

#Initialize LEDs
red = pyb.LED(1)
green = pyb.LED(2)
yellow = pyb.LED(3)
blue = pyb.LED(4)
red.on() #Indicates code began
pyb.delay(2000)

sw = pyb.Switch()
i2c = I2C('Y', freq=100000) #Set the frequency of the I2C and the 'Y' port of the pyboard

while True:

    pyb.wfi()   #Wait For Interrupt

    if sw():
        yellow.on() #File opened
        pyb.delay(500)
        log = open('/sd/IVTdata.csv', 'w')  #Open Excel file IVTdata to log the data
        log.write('Time(s),Current(mA),Cell 1(mV),Cell 2(mV),Cell 3(mV),Bat(mV),Pack(mV),Cell 1(0.1K),Cell 2(0.1K),Cell 3(0.1K)\n') #Logging header

        counter = 0
        while not sw():
            #Collect data
            blue.on()   #Indicator LED
            voltData = i2c.readfrom_mem(0x0b, 0x71, 13) #Extra bit to take the size designator for block commands
            current = i2c.readfrom_mem(0x0b, 0x0a,2)
            temp = i2c.readfrom_mem(0x0b, 0x72, 9)

            #Stores each unsigned short in a list format
            voltData = struct.unpack('<bHHHHHH', voltData)  #<H indicates little endian and unsigned short
            current = struct.unpack('<h', current)  #<h indicates little endian and signed short (current is signed)
            temp = struct.unpack('<bHHHH', temp)    #<b takes 1 byte, h or H takes 2 bytes

            log.write('{},{},{},{},{},{},{},{},{},{}\n'.format(counter,current[0],voltData[1],voltData[2],voltData[3],voltData[5],voltData[6],temp[2],temp[3],temp[4]))
            pyb.delay(500)  #Battery gauge refreshes registers once per second
            blue.off()
            pyb.delay(500)
            counter = counter + 1   #Time counter

        log.close()     #Close file after button is pressed
        yellow.off()    #File closed LED off
        pyb.delay(2000)
```

Figure 8. Data Logging Script

5 Data Collected

In addition to collecting the IVT data, some additional data was collected. A learning cycle was completed for the BQ40Z50 gauge to learn the exact characteristics of the batteries. The following were collected:

- Relative State of Charge (RSOC)
- True Full Charge Capacity (FCC)
- True Remaining Capacity
- Depth of Discharge (DOD) Passed Capacity

5.1 *IVT Data*

The discharge lasts approximately 18 minutes. The SOC begins at 100% and the flight ends at 10%, as reported by the BMS of the drone.

The current draw of the drone slowly increases to maintain constant power as the pack voltage begins to drop. The extreme spikes in the current and voltage are from either wind gusts or from flying the drone to different locations.

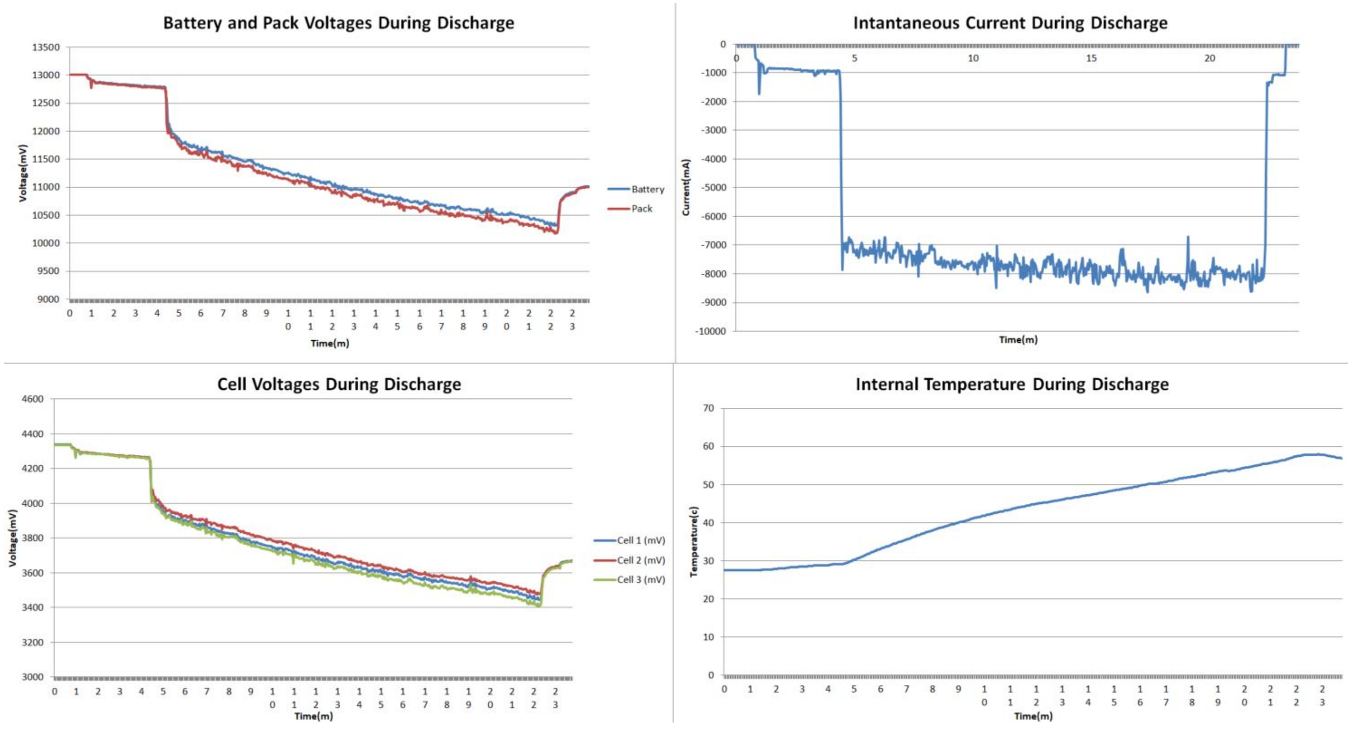


Figure 9. IVT Discharge Data

5.2 *Additional Data*

The additional data collected shows how the gauge reacts under a real load after completing a learning cycle. As can be seen in the RSOC graph, the gauge shows a smooth decline over the entire flight. The gauge is also able to predict the rest capacity that is regained after a discharge. True FCC changes quickly during discharge because of the increased current. The high current draw increases the IR losses within the battery. As the IR losses increase, the temperature also increases, further lowering the voltage curve of the battery during discharge. This means that the battery hits terminate voltage quicker than at a lower current draw, leading to a lower usable battery capacity.

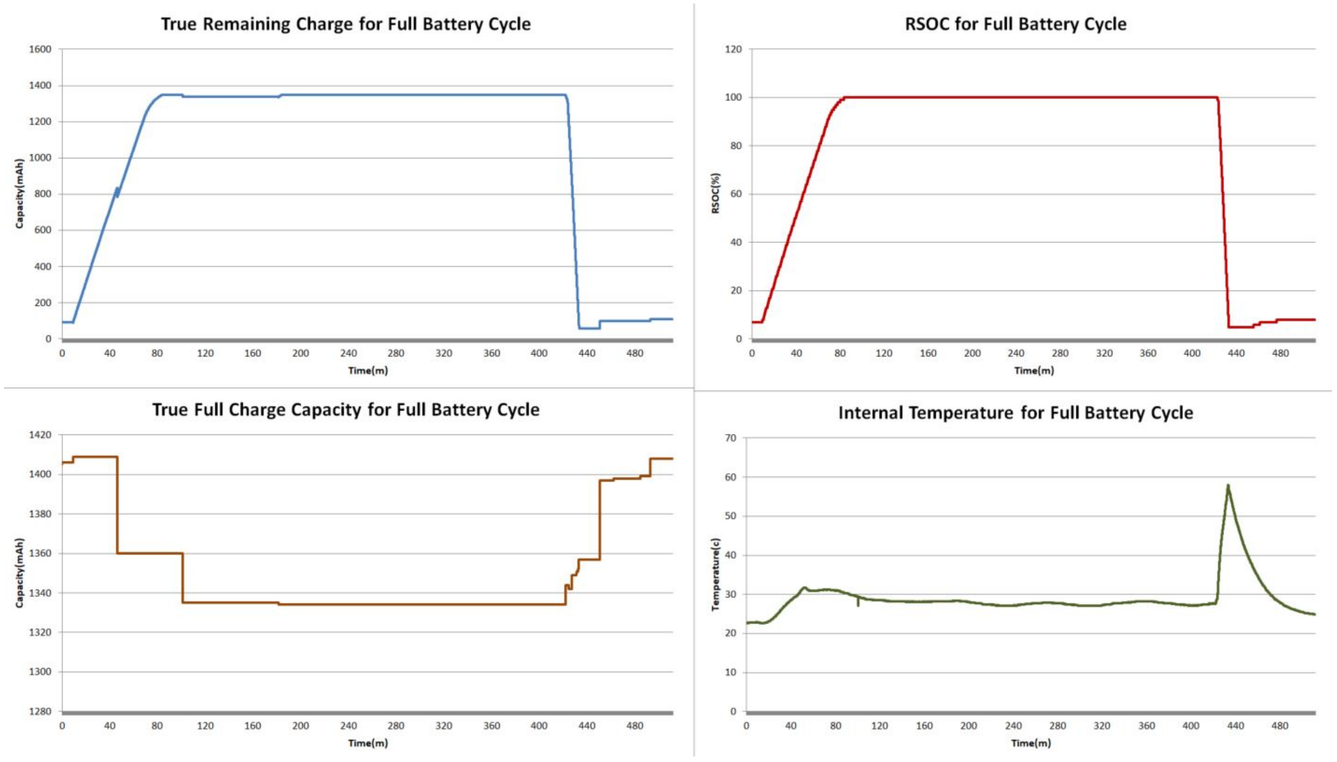


Figure 10. Additional Data Graphs

6 References

- I2C Communication with MicroPython, *class I2C – a two-wire serial protocol* (https://docs.micropython.org/en/latest/library/pyb.I2C.html#pyb-i2c)
- Logging Data to SD Card with MicroPython, *SDdatalogger* (http://wiki.micropython.org/SDdatalogger)
- Converting Bytes Into Usable Data, *Interpret Strings as Packed Binary Data* (https://docs.python.org/2/library/struct.html)
- Texas Instruments, *BQ40z50-R3 Technical Reference* (SLUUBU5)