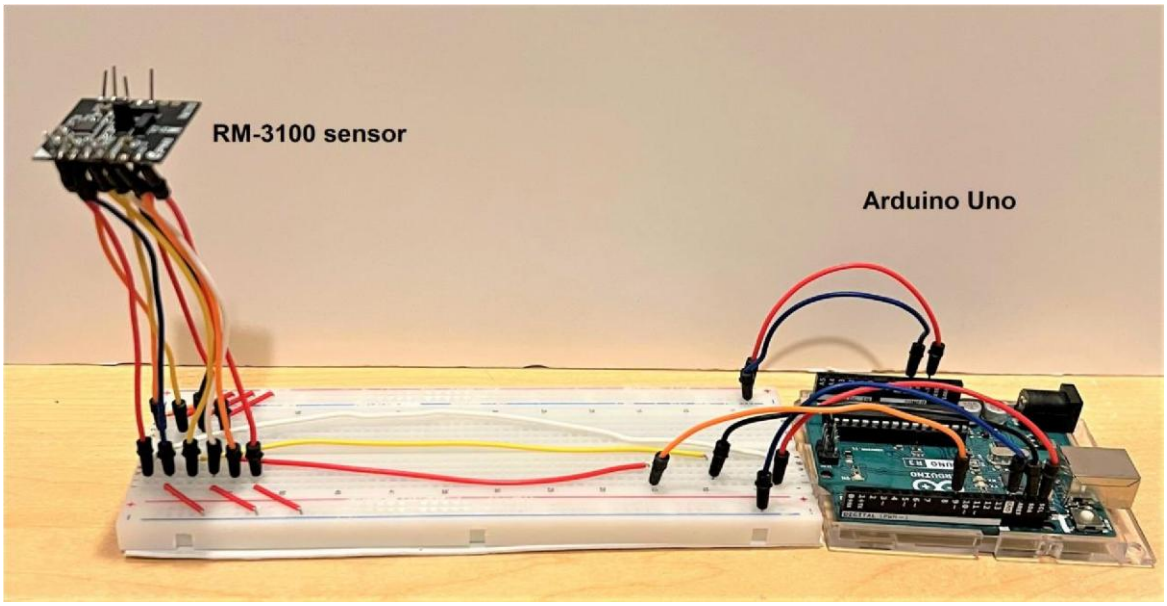
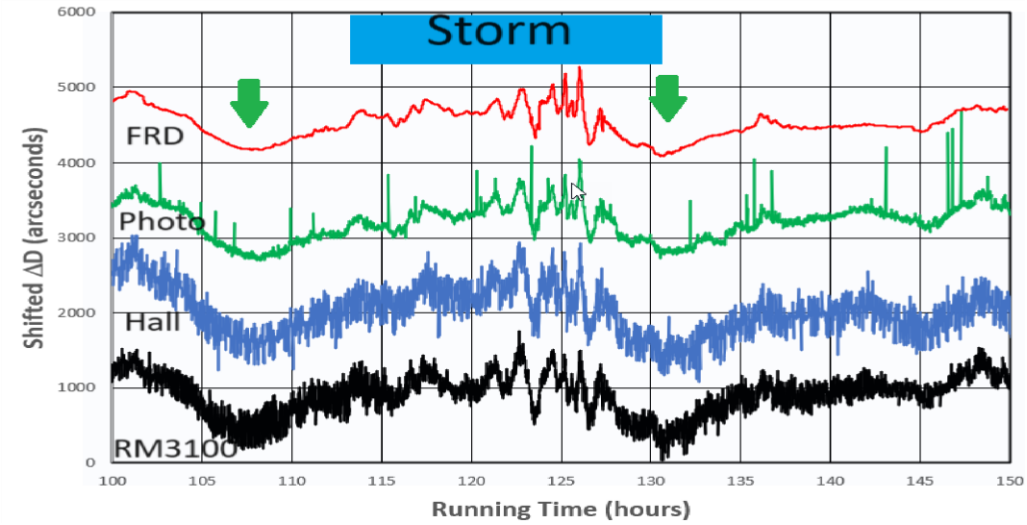
National Aeronautics and Space Administration



The Arduino Magnetometer for Space Weather Studies



The Arduino Magnetometer for Space Weather Studies

Developed by

Dr. Sten Odenwald (Astrophysicist)

Education alignment by

Dr. Hilarie Davis (Education Specialist)

Version 1.0 March 2025

NASA-Heliophysics Education Activation Team

NP-2023-5-072-GSFC

Introduction

Magnetometers are devices used to detect and measure magnetic fields. They range in simplicity from ordinary compasses that measure the local direction of the magnetic field polarity to sophisticated devices costing tens of thousands of dollars, which are used by scientists.

Magnetometers are important for studying space weather because some aspects of space weather involve sudden changes of Earth's magnetic field that occur during solar storms. These events can alter the strength of Earth's magnetic field at the ground by up to 5% and cause changes in the orientation of Earth's magnetic field by several degrees.

This Guide provides a step-by-step construction process for you to build your own RM3100 magnetometer capable of detecting these 'magnetic storms' during severe space weather events. The simple magnetometer design will cost less than $50 to build from ordinary items. For more information, contact Dr. Odenwald at *SpaceGuy598@gmail.com*

This Guide is a product of the NASA Heliophysics Education Activation Team, supported by NASA under cooperative agreement number NNH15ZDA004C.

Cover art: (Top) The Arduino microcomputer and the RM3100 sensor without covers and support. (Bottom) A sample of data from the Hall, Photocell and Arduino magnetometers compared to data from the Fredericksburg Magnetic Observatory (FRD) showing sensitivity of these systems as compared to professional-grade instruments for the geomagnetic storm during June 2024. The green arrows indicate the diurnal dips for the Sq current effect. (Credit: The Author).

Table of Contents

Part I: Notes for Educators

NASA space missions often require measuring magnetism on the Sun, on Earth, and on other planets and bodies in our solar system. ***Heliophysics*** is the study of the Sun and its effects on Earth and the solar system. In this guide, students will learn how Earth's magnetic field interacts with the solar wind and keeps Earth safe and how studying magnetism can help scientists learn about the unique environment that the Sun creates in the solar system.

When your students use this guide, the following information will provide an educational context for its use. The project description includes information about the Next Generation Science Standards that apply and provides guiding questions and an assessment to help teachers gauge student performance in constructing the device, acquiring data, and interpreting the data.

1. Overview

Students will measure Earth's changing magnetic field during geomagnetic storms caused by increased solar activity. The Sun goes through an 11-year sunspot cycle with periods of increased numbers of sunspots that are related to the frequency of solar flares and other 'solar storms', which can affect Earth's magnetic field. Scientists refer to the effects of these solar storms as 'space weather.' The strongest storms occur during and just after a period in the Sun's cycle called Solar Maximum. Currently our Sun is in its fourth year of sunspot cycle number 25 with a maximum predicted to occur in 2025. Storms that are strong enough to be detected by smartphones only occur a few times each month during the time of peak solar activity. Be sure to check where the Sun is in its cycle before attempting this experiment with students. Use a service such as the one provided by the NOAA Space Weather Prediction Center (https://swpc.noaa.gov) to see if a storm is occurring, or when the next one may arrive.

2. Objective

Students will be able to observe space weather phenomena that cause variation in Earth's magnetic field.

3. Explanation

Compared to professional magnetometers used at magnetic observatories, most of the simple designs such as soda bottle magnetometers are not sensitive enough to detect weak geomagnetic storms with Kp<7, but they can be used to detect some of the stronger storms. The process requires careful analysis of the data. For severe storms with Kp>8,

these events should be detectable in most locations across North America. However, these storm events are rare and occur about once every few months during times when the Sun is active (called sunspot maximum). They are unpredictable, so you need to carefully monitor such space weather websites such as SpaceWeather.com to see if a storm is likely in the next 24-48 hours.

4. Assessment

Use the answers to the questions during data analysis to determine if students can accurately collect and analyze data during a geomagnetic storm. These questions can include:

- What kinds of solar events can cause Earth's magnetic field to vary?
- Why are compass needles affected by solar storms?
- How does a magnetometer detect changes in Earth's magnetic field?
- What property of Earth's magnetic field is being measured by the magnetometer?
- What is the typical range of measurements that you detect during a strong storm?

5. Targeted High School NGSS Standards

Appropriate for designs involving Hall sensors, photocells, smartphones, and Arduino.

HS-ETS1-2 Design a solution to a complex real-world problem by breaking it down into smaller, more manageable problems that can be solved through engineering.

HS-PS3-5 Develop and use a model of two objects interacting through electric or magnetic fields to illustrate the forces between objects and the changes in energy of the objects due to the interaction.

6. A Glossary of Terms

Boom – a mechanical device on a spacecraft that keeps certain sensitive instruments far from the spacecraft to reduce interference.

Current – a flow of charged particles such as electrons and is measured in units called amperes

Dynamo – a device containing a rotating magnet that produces electrical currents

Electromagnetic – something that has both electrical and magnetic properties

Field – an influence, usually a force, that exists in the space surrounding an object

Force – an influence that causes nearby or distant objects to move, sometimes without physical contact

Gauss – a unit of measurement for magnetism in a system of units that also uses centimeters and grams

Interstellar – literally the space between stars, usually occupied by various gases and clouds of dust

Magnetometer – an instrument for measuring the intensity and direction of a magnetic field

Polarity – the direction of a force or current such as magnetism (North or South-type) or on a battery (positive or negative)

Spacecraft – a platform carried into space that contains a collection of instruments for measuring distant objects and environments in space

Space weather – a collection of phenomena that describe how Earth and the other planets respond to solar activity

Sunspot – a dark spot in the solar surface where magnetic fields very intense causing the gas to be cooler and emit less light making it dark compared to the sun's bright surface

Tesla – a unit of measurement for magnetism in a system that uses meters and kilograms; one Tesla equals 10,000 Gauss. As an example, the magnetic field at Earth's surface is about 50 microTeslas (0.00005 Tesla) or 0.5 Gauss in strength. In this document we use microTesla (μT) and nanoTesla (nT). There are 1000 nT for each μT.

Vector – a quantity that is defined both by its amount and its direction. The motion of a body is defined by its velocity vector, which has an amount (called speed) and a direction (up, down, etc.).

Part II. An Arduino-based, high-sensitivity magnetometer

1. Background

In recent years, several new magnetometer sensors have become commercially available that span the gap between the inexpensive Hall Effect designs we have previously investigated with sensitivities above 100 nT, and the expensive fluxgate magnetometer systems that work below 1 nT. One of these intermediate-cost systems is based on the RM3100, a three-axis sensor developed by George Hsu and a team of engineers at PNI Sensor Corporation in Santa Rosa, California in 2012. It has a sensitivity 10-times greater than Hall sensors and is a magneto-inductive device.

The operating principle involves measurement of the time it takes to charge and discharge an inductor between an upper and lower threshold by means of what is called a Schmitt trigger oscillator. This time is proportional to the applied magnetic field strength within a specified operational range.[1]

According to the PNI RM3100 Testing Boards Manual, "*Each sensor coil of the RM3100 serves as the inductive element in a simple LR (inductance-resistance) relaxation oscillation circuit, where the coil's effective inductance is proportional to the magnetic field parallel to the sensor axis. The LR circuit is driven by the MagI2C ASIC, and the MagI2C's internal clock is used to measure the circuit's oscillation frequency, and hence the magnetic field. Since the RM3100 works in the frequency domain, resolution and noise are established cleanly by the number of MagI2C internal clock counts (cycle counts). In comparison, fluxgate and [Hall Effect] technologies require expensive and complex signal processing to obtain similar resolution and noise. Also, the output from the MagI2C is inherently digital and can be fed directly into a microprocessor, eliminating the need for signal conditioning or an analog/digital interface between the sensor and a microprocessor. The simplicity of PNI's geomagnetic sensor, combined with the lack of signal conditioning, makes it easier and less expensive to implement than alternative fluxgate or magneto-resistive (Hall Effect) technologies.*"

The RM3100 sensor technology is being used by HamSci.org to create a network of inexpensive magnetometer stations operated by Ham radio operators and other interested individuals[2]. It is also being used by the NASA Electrojet Zeeman Imaging

[1] https://d-nb.info/1148110194/34

[2] https://hamsci.org/mag_install

Explorer (EZIE), whose educators are creating a network of these sensors called EZIEMag[3] to study geomagnetic storms in the classroom.

How it works

The heart of the magnetometer is the sensor itself, which is available from PNI for under $30.00 and shown in Figure 1. The output from the sensor cannot be measured with a common multimeter as for previous designs.

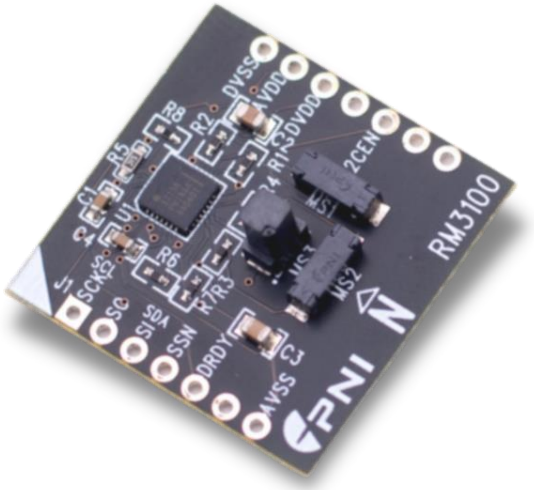


Figure 1. The RM3100, three-axis magnetometer.

Instead, the output from the sensors must be controlled by a microprocessor. The RM3100 circuit board uses both the I2C and SPI input/output communication protocols for interfacing with an inexpensive Arduino microcontroller like the one shown in Figure 2.

[3] https://learninglab.si.edu/collections/ezie-mag-steam-challenge-1-your-aurora-storybookexibition/0jnvICVNmzQFpiYh

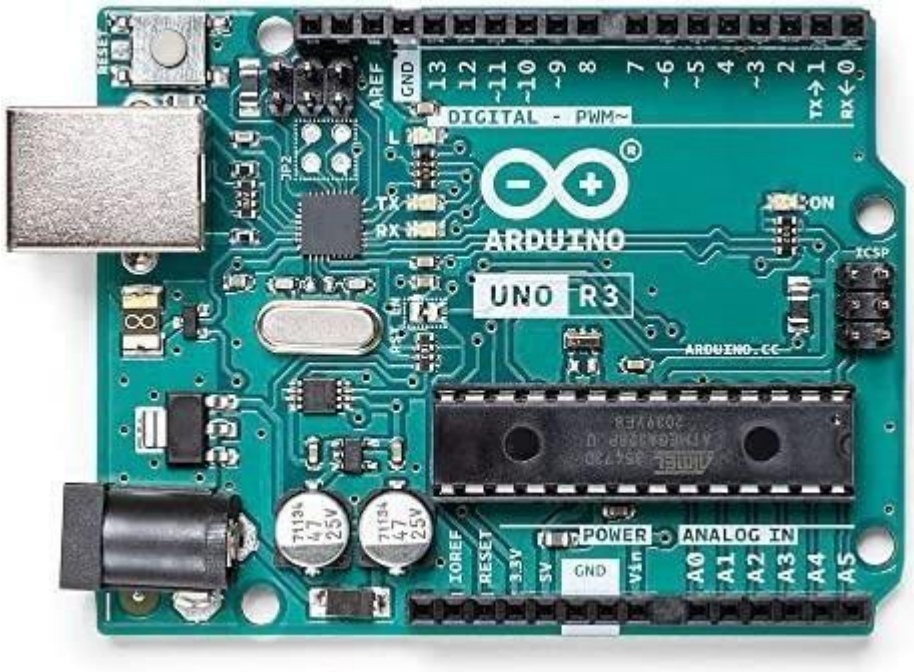


Figure 2. The Arduino 'Uno' microcontroller board.

The Arduino Uno board is a very popular microcontroller used by amateurs who design robotic or other devices controlled by computer commands. The computer software sends digital commands to the Arduino controller, which then passes the data back to the computer on the I2C interface. Arduino UNO Rev3 is the most used and documented board in the world. It has 14 digital input/output pins, 6 analog inputs, a USB connection, a power jack, and a reset button. Arduino is an open-source hardware, software, and content platform with a worldwide community of over 30 million active users.

Once the proper connections are made, the Arduino is connected via a USB cable to an external computer or laptop that is running the Arduino software. There are several versions of this downloadable code that are available[4] including GitHub[5].

The RM3100 is currently being deployed in the HAMSCI citizens science project, using code developed by HAMSCI members with differential i2c signaling using CAT5-6 cabling over distances to at least 150 meters. They also developed a mounting technique for the project using common PVC water pipe and fittings to place the RM3100 below ground surface for passive temperature stabilization to ±1°C. Using cycle counts of 400,

[4] https://forum.arduino.cc/t/spi-communication-with-rm3100-magnetometer-testing-board/702040

[5] https://github.com/hnguy169/RM3100-Arduino

with a 1-per-second sample rate, resolution of 5nT is readily achievable. Other designers have developed alternative Arduino-based systems[7] and coding, and have achieved similar sensitivities between 6 and 30 nT.

Total Cost: $82.00

$35.00 RM3100
$28.50 Arduino Uno
$8.58 USB Interface cable
$9.59 PC board, jumpers.

2. Materials

- **RM3100 breakout board**; $35.00 including shipping; https://www.pnicorp.com/product/rm3100-breakout-board/
- **Arduino Uno** – $28.50 https://tinyurl.com/2p9fbbsn - Note: Make sure it is marked with the 'Arduino' trademark otherwise it is likely to be a cheap copy that may not work.
- **Arduino cable** - $8.58; 5-feet
- **Solderless PC board, power supply and jumper cables** - DGZZI 1Set Electronics Fun Kit(1PCS Power Supply Module + 1PCS Solderless 830 tie-Points Breadboard + 65pcs Jumper Wire) for Arduino; $9.59
- **Soldering iron and solder.**
- **PC board jumpers**

3. Procedure

Arduino set-up

Step 1) First we must install the *Arduino IDE* software environment where you will be writing the code to interact with the RM3100 via the Arduino interface board. Visit the 'Getting Started with Arduino Products' website[6] and follow the directions for downloading the software[7] and installing it on your laptop or computer. At the download page it will ask for a small donation, or you can skip this and 'just download'. The .exe file for the Windows platform is about 150 megabytes. Click on the file and 'open' it. *Arduino Ide* will

[6] https://www.arduino.cc/en/Guide
[7] https://www.arduino.cc/en/software

install into your Program folder. Your system's security software may block the Installer from downloading device drivers so you will have to click on the requisite boxes to permit the 'unknown program' to access your computer. The *Arduino IDE* program will then start up and show the screen in Figure 3. The *Arduino IDE* icon will automatically appear on your desktop window after installation.

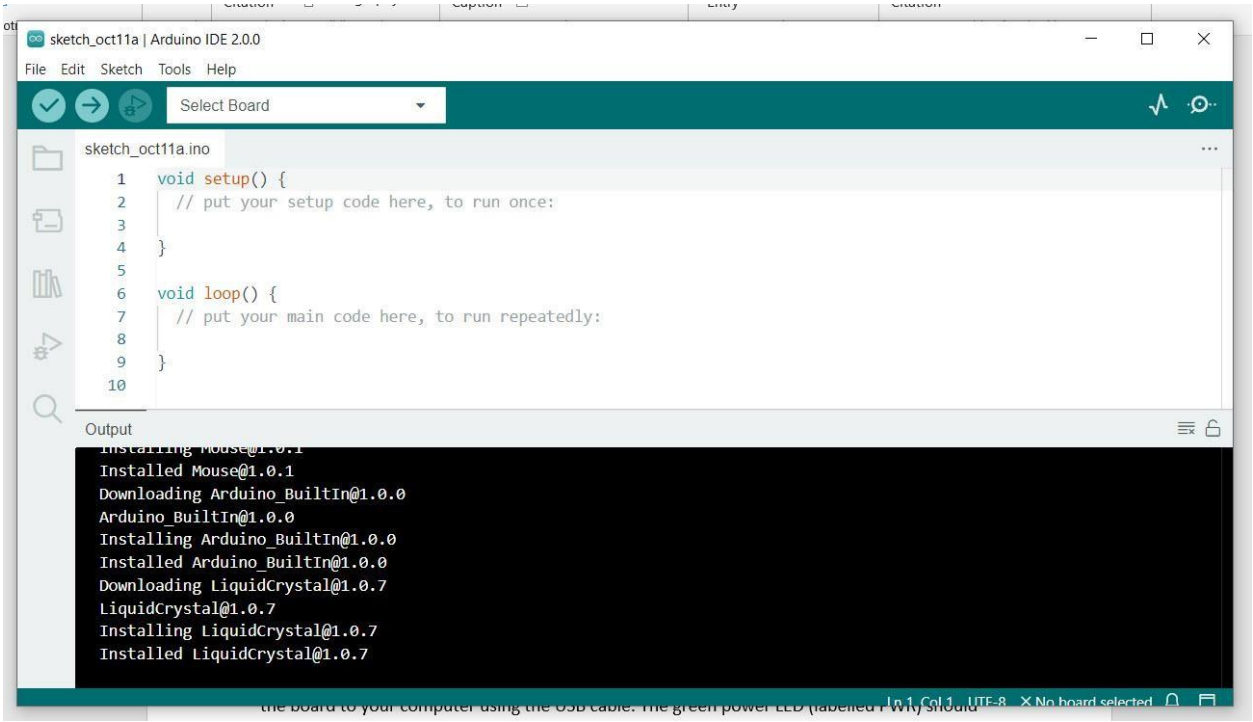


Figure 3. Example of the Arduino IDE startup window. Top area is where you will write the C-language code, and the bottom window is the output from the program code after it is compiled and executed.

Step 2) The Uno automatically draws power from either the USB, an external 9V battery, or USB power supply. Connect the board to your computer using the USB cable. The green power LED (labelled PWR) should go on.

Step 3) Exit IDE and re-start the program by clicking on the icon on your desktop. The program window will open and, with the Arduino plugged into the USB port and active (PWR LED on), the IDE program will automatically detect the Arduino board as an active device for purposes of running code that you develop. With the program running, click on the 'Tools' tab and then 'Port'. A popup window will tell you which COM port was assigned to the Arduino Uno e.g. 'COM4 (Arduino Uno)'. In the window on the top bar use the down arrow to select the COM port being used by Arduino. The bar will then indicate 'Arduino Uno'. This means that the program has properly identified that the Arduino is now linked as an I/O device for the program.

Step 4) Read through the '*Getting Started with Arduino IDE 2.0*' guide[8] and familiarize yourself with the basic command structure and how to read data and load programs to the microprocessor. Here are some basics:

- Your sketchbook is where your code files are stored. Arduino sketches are saved as .ino files and must be stored in a folder of the exact same name. For example, a sketch named *my_sketch.ino* must be stored in a folder named *my_sketch*. Typically, your sketches are saved in a folder named Arduino in your Documents folder. To access your sketchbook, click on the folder icon located in the sidebar of the IDE window.
- All commands and functions are case-sensitive.
- All lines of code terminate with a semi-colon.
- A comment line can be added by preceding it with //
- Arduino memory can only store a few hundred lines of code so be frugal with unnecessary comment lines.

Step 5) Practice using the IDE environment by writing simple code. HackerEarth.com has a great Tutorial on writing simple code for the Arduino[9]. Note: you cannot cut and paste code from a document into the code area because hidden control characters (line feeds etc.) are not Arduino commands or command line terminators, and will cause an '*stray '\342' in program*' error. Always re-type each line of code by hand into the code window or copy the code from a pure ascii .txt file. In Figure 4, the IDE window has an upper area

```
void setup() {
}
```

and a lower area

```
void loop () {
}
```

The *setup()* function only runs once and is used to initialize the pin modes and start serial communication. This function must be included even if there are no statements to execute. For example, here is a simple program:

```
void setup() {
// Here is my first program;
Serial.begin(9600);
Serial.println("Hello world....NASA rocks the universe!");
}
```

[8] https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2

[9] https://www.hackerearth.com/blog/developers/arduino-programming-for-beginners/

```
Void loop () {
}
```

With the 'Tools' Board set as 'Arduino Uno' and the 'Tools Port' set at COM4, execute the program by clicking on the Right Arrow in the top bar. The program will upload and compile. If you click on the Serial Monitor icon, the lower window will show the Serial Monitor area which will say '*Hello World...NASA rocks the universe!*' as shown in Figure 4. Note that all comments start with '//' .

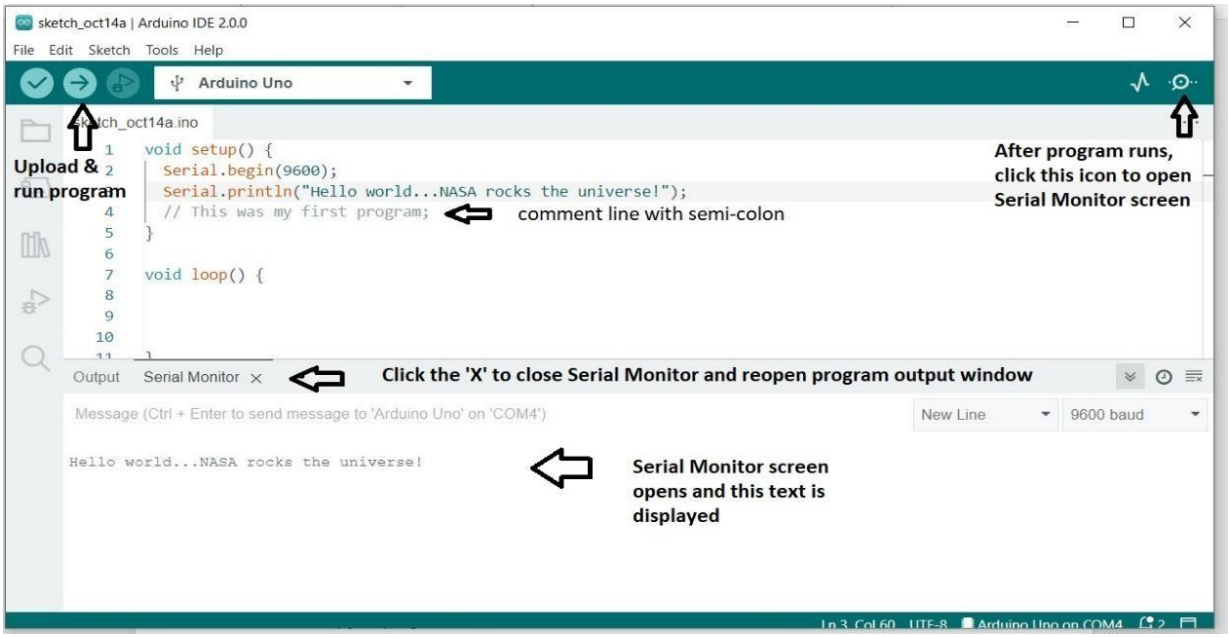


Figure 4. Screen shot of IDE work area.

Note that after the program compiles it tells you how much space the code occupied: *"Sketch uses 1512 bytes (4%) of program storage space. Maximum is 32256 bytes. Global variables use 226 bytes (11%) of dynamic memory, leaving 1822 bytes for local variables. Maximum is 2048 bytes. "*

The maximum program storage is 32,256 bytes so when you write code you have to be careful to write it efficiently.

Step 6) To actually DO something with the Arduino interface we can, for example, program an LED to flash on and off. The setup looks like Figure 5. You will need an LED and a 220-Ohm resistor, along with a breadboard.

- Connect the Arduino to the Windows system via a USB cable
- Connect the 13th digital pin of Arduino to the positive power rail of the breadboard and GND to the negative

- Connect the positive power rail to the terminal strip via a 220-ohm resistor
- Fix the LED to the ports below the resistor connection in the terminal strip
- Close the circuit by connecting the cathode (the short chord) of the LED to the negative power strip of the breadboard

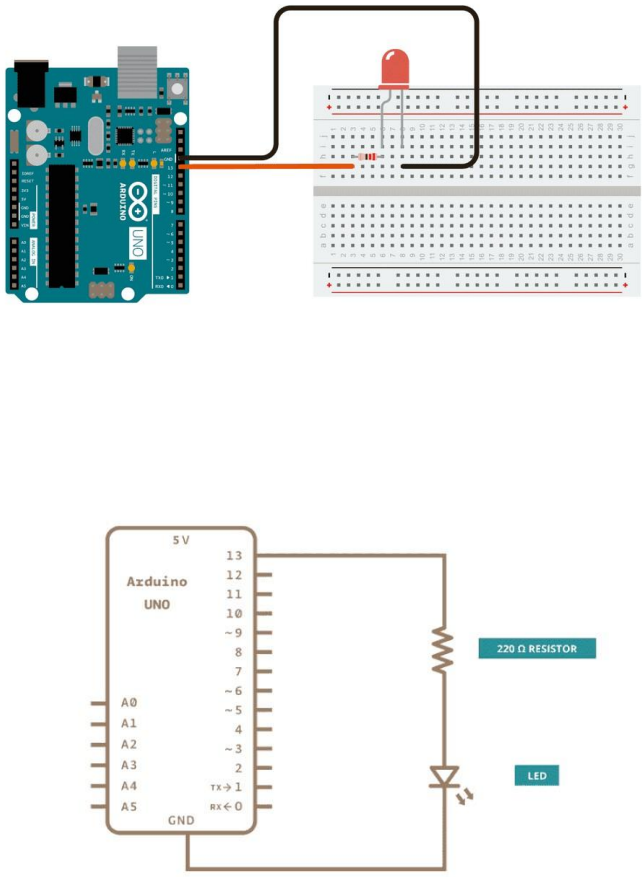


Figure 5. Setup for flashing LED

The program code in the IDE upper window will be

```
                    void setup ( )
                    {
                    pinMode (13, OUTPUT);        //pin 13 is set as output pin
                    }
                    void loop( )
                    {
                digitalWrite (13,HIGH);        // Turn ON the LED on pin 13
        delay (1000);                    //Wait for 1sec                    digitalWrite
(13, LOW);       //Turn OFF the LED on pin 13                delay (1000);
//Wait for 1 second
```

}

In the setup block, Pin-13 is assigned as the output from the controller to the LED. The *loop()* program writes to Pin-13 a high-voltage +5.0V pulse to turn on the LED. The next line has the program wait for 1000 cycles using the *delay(1000)* function and then executes the write to Pin-13 to put the voltage in a LOW 0-volt state to turn the LED off. If it is a new program, use the Tools to again select the Board as Arduino Uno. When you then click the 'upload' icon in the IDE workspace, this 10-line program executes and causes the LED to blink on for one second and then off for one second.

You will notice that the Arduino keeps on blinking even when you exit the IDE program. That's because you uploaded the program to the Arduino, and the microcontroller is now operating autonomously! If you provided it with a 9V battery power supply, you can disconnect the USB cable and the program will run without connection to the computer. Even if you unplug the Arduino from your laptop, when you plug it back in it will reload the program and keep blinking. To stop the program once-and-for-all, upload a new program such as the 'Hello World' program. It will over-write the blinking program and stop its execution.

Step 7) There are many different commands in the Arduino language. It would be impossible to review them all in this guide. Instead, we will focus on the job at hand, which is to interface the Arduino with the RM3100 magnetometer. Once the electrical connections are made, the program code (called a 'sketch' in Arduino-lingo) can be copied into the IDE program area and uploaded to test the RM3100.

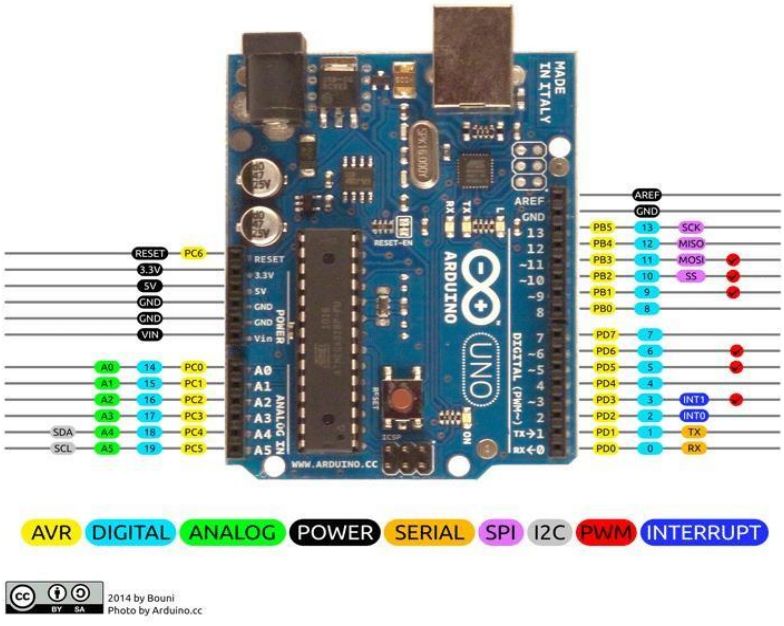


Figure 6. Arduino Uno board pin identifications. Note location of 3.3V and GND pins. These will be used for powering the RM3100 installed on the bread board.

Installation of the Power Supply.

Step 8) The Arduino Uno board includes one each of both 3.3V (called '3V3') and 5-Volt regulated power supplies. Figure 6 shows the various pins on the Arduino Uno board. The 3.3 and 5Volt + pins are shown in black together with the accompanying ground GND (-) pins. Place the Arduino board against the PC board end and connect the jumpers from the GND pin to a hole in the (-) power rail, and the 3V3 pin to the adjacent (+) power rail of the PC board making sure to keep the polarities correct as shown in Figure 7.

Step 9) On the multimeter, clip the common and input leads to two jumpers with pins that can be inserted into the PC board. These pins are also the same size for the Arduino board. Note that the 'Input' voltmeter line must go to the '3V3' pin, and the 'Common' voltmeter line must go to the GND pin to get correct voltage readings. Ideally, the output voltages from the 3V3 rail on the board should be very close to 3.3-volts.

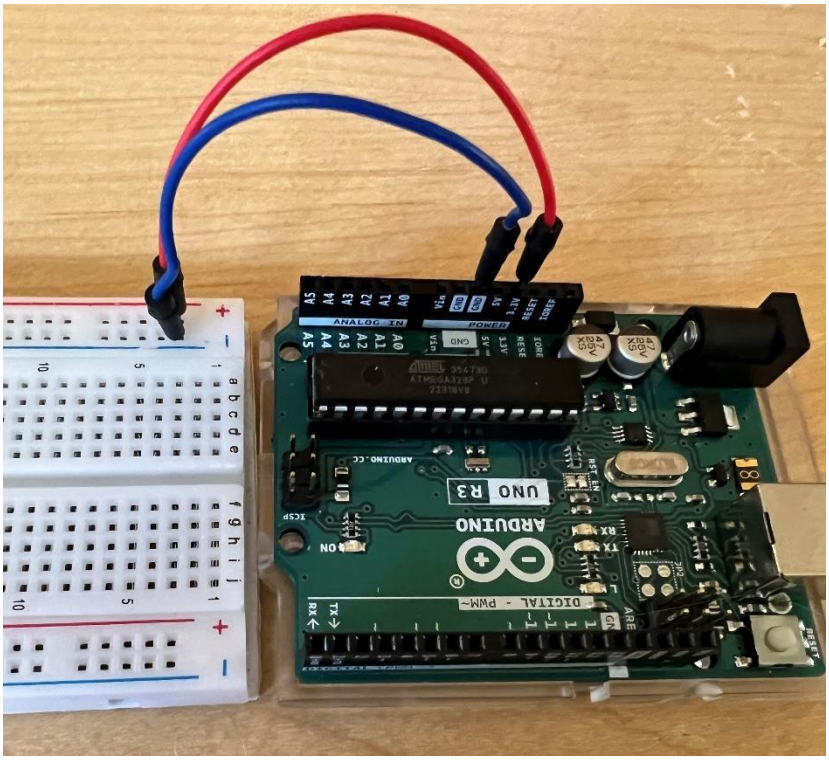


Figure 7. Arduino Uno board (right) and PC board (left). The red jumper (+3.3-V) connects to the top rail +pin and the blue jumper (GND) connects to the adjacent - rail.

Thinking ahead to the installation of the RM3100, we must decide which of the two communications protocols we will be using: I2C or SPI. The I2C digital interface will be used because a digital signal connection is potentially much more robust to noise from interference.

Connecting the RM3100 to the Arduino

Figure 8 shows the pin assignments for the RM3100 board using the I2C interface. A step-by-step hookup guide is available at *github.com*[10]. The pin assignments are shown in Table 4. A diagram showing the connections is shown in Figure 9.

Figure 8. The RM3100 development board with the three magnetometers; one for each axis.

Table 4. Pin assignments for the I2C protocol.

RM3100 Board			**Arduino Uno I2C**
Pin #	**Pin Name**	**Description**	**Pin #**
1	SCK/SCL	I2C-Serial Clock Line	D15 - SCL
2	S0	I2C – Bit 1 of slave address	GND
3	SI/SDA	I2C – Serial Data Line	D14 - SDA
4	SSN	I2C – Bit 0 of slave address	GND
5	DRDY	Status Line	D9
7	AVSS	Ground pin for analog section	GND
10	I2CEN	I2C enable pin (set HIGH)	AVDD +3V3
12	DVDD	Supply voltage: digital section	AVDD +3V3
13	AVDD	Supply voltage: Analog section	AVDD +3V3
14	DVSS	Ground pin for digital section	GND
6,8,9,11		Do not connect	

[10] at https://github.com/hnguy169/RM3100-Arduino/blob/main/RM3100%20Arduino%20Quick%20Guide.pdf

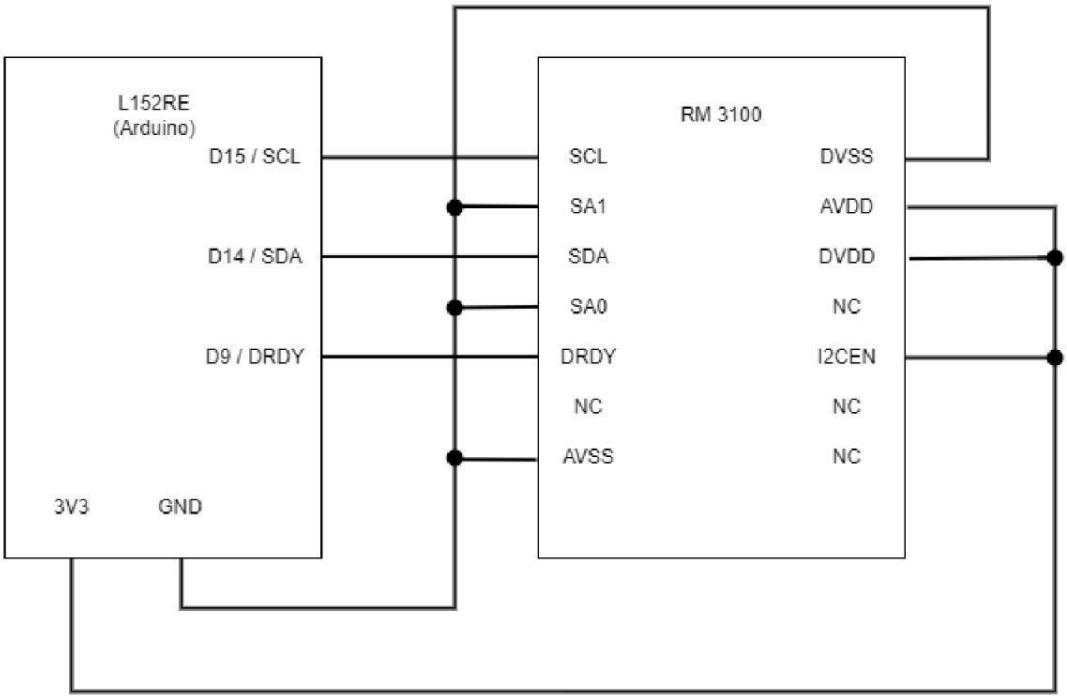


Figure 9. Connections between Arduino and RM3100 boards. Note that the 3.3-volt (3v3) power is used.

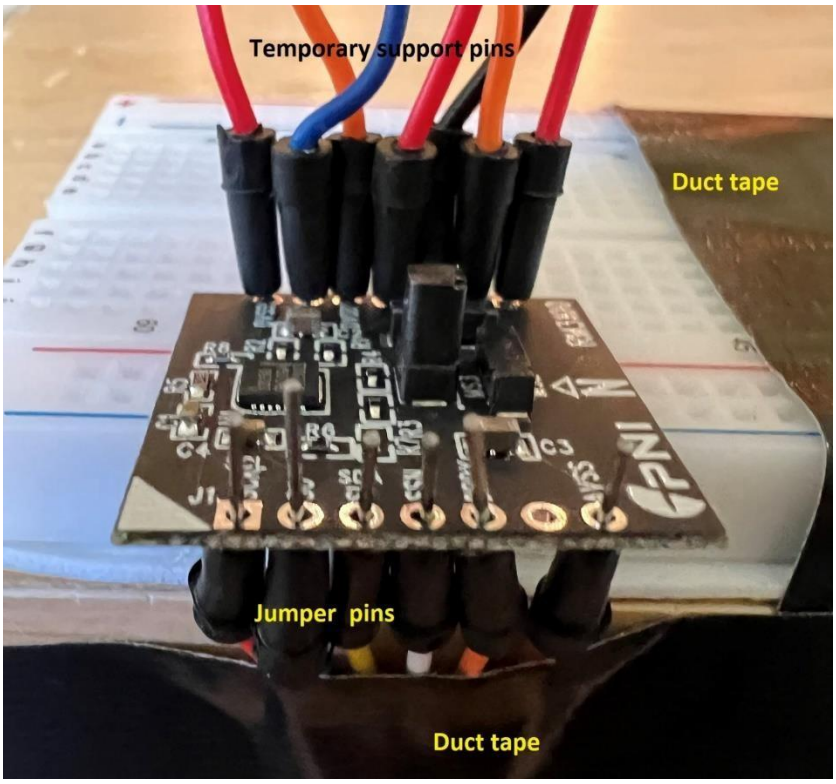


Figure 10. Example of the soldering jig for the RM3100 board. The temporary support pins are used to attach the RB3100 board to the main board for support only.

The RM3100 board does not come with pins to insert into the breadboard.

Step 10) Select 10 short jumpers (with pins on both ends of course). These should be about 9 cm long from tip-to-tip. If shorter jumpers are available with lengths of 4 cm these would be ideal.

Step 11) Figure 10 shows a temporary jig for keeping the jumper pins and RM3100 board secure while you solder the terminals. The RM3100 board is placed at the edge of the white PC board and the holes are aligned with the PC board pin holes. Jumpers are attached through the RM3100 holes into the white PC board, and a second set of jumpers are inserted from the bottom through the foreground holes and taped into place with a strip of duct tape.

Step 12) Carefully solder the jumpers into place. Repeat this process for the second row of connections on the opposite side of the RM3100 board.

Step 13) Plug the jumpers into the main PC board. The RM3100 board will be suspended about 4 to 6 cm above the board surface. Once we have confirmed that the software works, we will create a more permanent support for the RM3100 board.

Step 14) Follow the wiring diagram in Figure 9 and Table 2 and connect the RM3100 board to the Arduino board. Figure 11 shows an example of this.

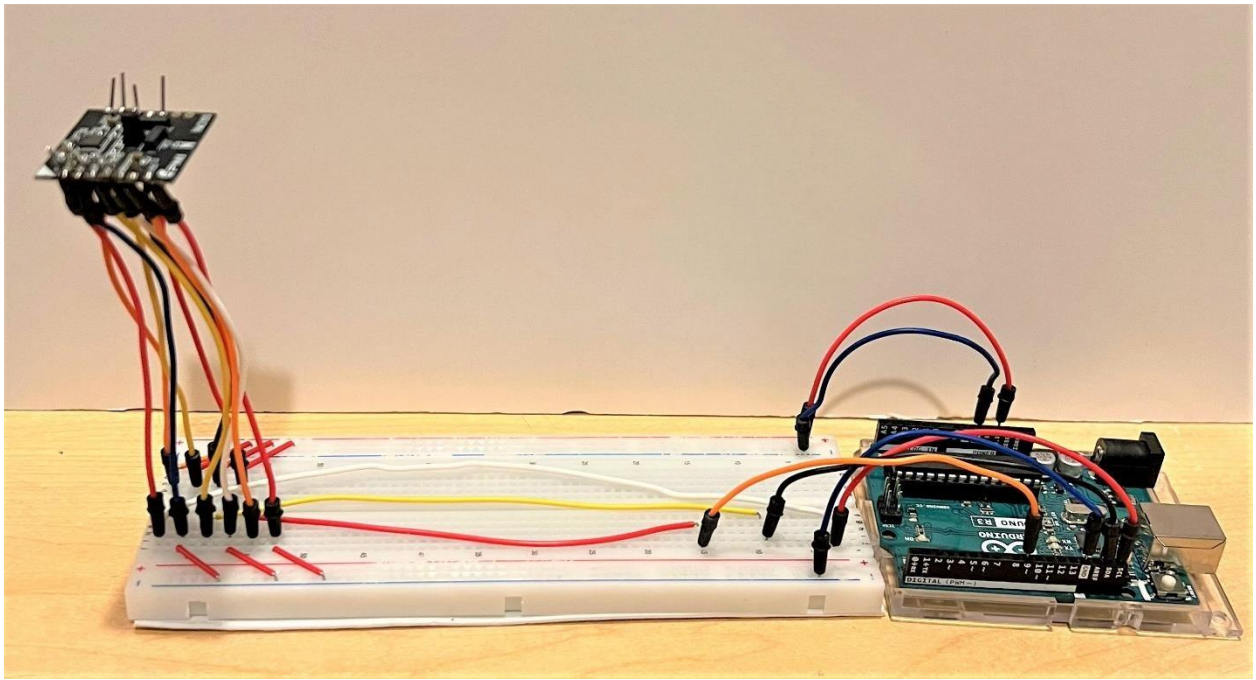


Figure 11. The Arduino (right) and RM3100 (left) boards connected with jumpers.

Creating an enclosure for the magnetometer

The magnetometer chip is suspended above the board by eight wires and the breadboard is exposed. This is not an ideal setup for this precision system, so it needs to be stabilized and enclosed. The simplest way to do this is with foam board. The details will vary depending on your specific set up and dimensions, but Figures 12 and 13 show one configuration. The magnetometer chip has been carefully hot glued by two adjacent edges to the top of the corner of the vertical tower.

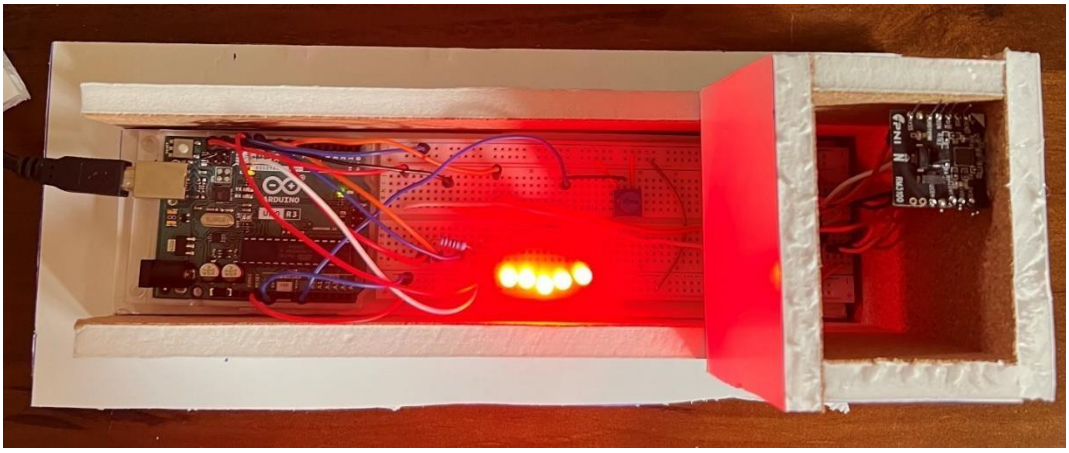


Figure 12. The magnetometer box seen from above. Note the mounting of the magnetometer chip in the upper-right corner.

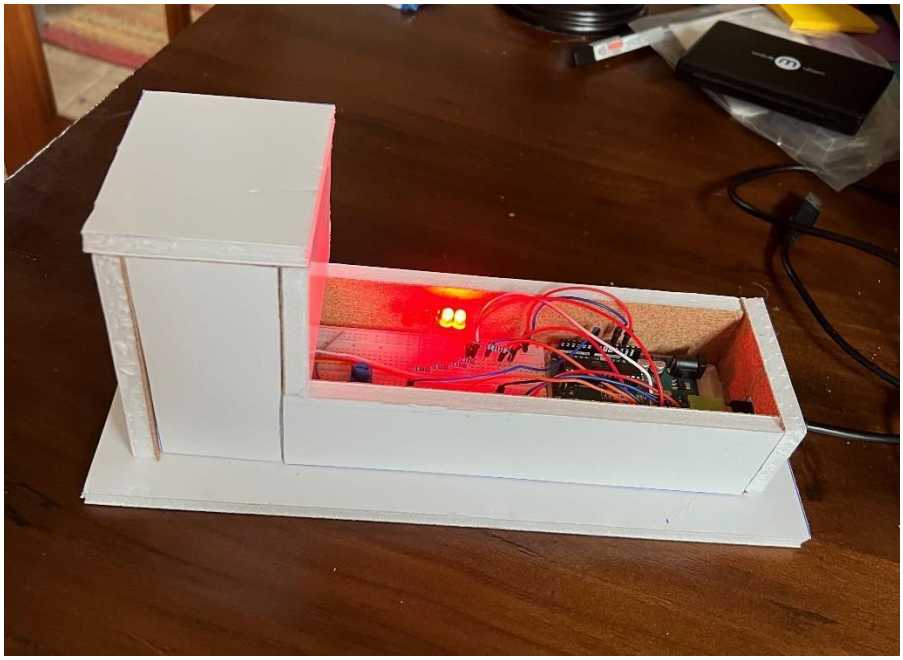


Figure 13. Side view of magnetometer box.

Loading and testing the code.

Step 15) In Arduino programming, a complete program is called a 'sketch'. PNI created a sample sketch for the Arduino Uno I2C[11] protocol that tests out the various features of the RM3100. The sketch is shown at github.com[12]. Click on the 'Raw' button to reveal the code in pure ASCII form. The code between lines 26 and 35 was originally developed for a different microprocessor and looks like this:

```
//Enable Pullup Resistors on Nucleo-L152RE I2C Pins (Pins PB_8/D14 and PB_9/D15) const
PinMap PinMap_I2C_SDA[] = {
 {PB_9,  I2C1, STM_PIN_DATA(STM_MODE_AF_OD, GPIO_PULLUP, GPIO_AF4_I2C1)},
 {NC,    NP,    0}
};

const PinMap PinMap_I2C_SCL[] = {
 {PB_8,  I2C1, STM_PIN_DATA(STM_MODE_AF_OD, GPIO_PULLUP, GPIO_AF4_I2C1)},
 {NC,    NP,    0}
};
```

This code block must be deleted and replaced by the two lines highlighted below. The purpose of this code is to enable a 'pull-up' resistor in the Arduino Uno[17]. The corrected sketch looks like this:

```
#include <Arduino.h>
#include <Wire.h>
#define RM3100Address 0x20 // RM3100 slave address: Pin 2 and 4 set to LOW
//pin definitions
#define PIN_DRDY 9 //Set pin D8 to be the Data Ready Pin
//internal register values without the R/W bit
#define RM3100_REVID_REG 0x36 // Hexadecimal address: Revid internal register
#define RM3100_POLL_REG 0x00 // Hexadecimal address:Poll internal register
#define RM3100_CMM_REG 0x01 // Hexadecimal address: CMM internal register
#define RM3100_STATUS_REG 0x34 //Hexadecimal address:Status internal register
#define RM3100_CCX1_REG 0x04 // Hexadecimal address: Cycle Count X1 register #define
RM3100_CCX0_REG 0x05 // Hexadecimal address: Cycle Count X0 register //options
#define initialCC 200 // Set the cycle count to 200
#define singleMode 0 //0 =continuous measurement; 1 = single measurement
#define useDRDYPin 1 //0 = not using DRDYPin ; 1 = using DRDYPin to for
data uint8_t revid; uint16_t cycleCount; float gain;
```

[11] https://github.com/hnguy169/RM3100-Arduino/blob/main/RM3100%20Arduino%20Quick%20Guide.pdf

[12] https://github.com/hnguy169/RM3100-Arduino/blob/main/RM3100_Arduino_I2C/RM3100_Arduino_I2C.ino [17] https://www.arduino.cc/en/Tutorial/BuiltInExamples/InputPullupSerial

```
void setup() {
  pinMode(PIN_DRDY, INPUT); pinMode(SCL,INPUT_PULLUP);
//enable SCL pull-up resistor
pinMode(SDA,INPUT_PULLUP);  //enable SDA pull-up resistor
  Wire.begin(); // Initiate the Wire library
Serial.begin(9600); //set baud rate to 9600
  delay(100); // set delay to 0.1 seconds or 100 milliSeconds
revid = readReg(RM3100_REVID_REG);
  Serial.print("REVID ID = 0x"); //REVID ID should be 0x22
Serial.println(revid, HEX);
  changeCycleCount(initialCC); //change the cycle count; default = 200
  cycleCount = readReg(RM3100_CCX1_REG);
  cycleCount = (cycleCount << 8) | readReg(RM3100_CCX0_REG);
  Serial.print("Cycle Counts = "); //display cycle count
  Serial.println(cycleCount);
  //linear equation to calculate the gain from cycle count
  gain = (0.3671 * (float)cycleCount) + 1.5;
  Serial.print("Gain = "); //display gain; default gain should be around 75 for the default cycle count of 200
  Serial.println(gain);
if (singleMode){
    //set up single measurement mode
writeReg(RM3100_CMM_REG, 0);
    writeReg(RM3100_POLL_REG, 0x70);
  }
  else{
    // Enable transmission to take continuous measurement with Alarm functions off
writeReg(RM3100_CMM_REG, 0x79);
  }
}
void loop() {
long x = 0;   long
y = 0;   long z =
0;
  uint8_t x2,x1,x0,y2,y1,y0,z2,z1,z0;
  //wait until data is ready using 1 of two methods (chosen in Options)
if(useDRDYPin){
    while(digitalRead(PIN_DRDY) == LOW); //check RDRY pin
  }
  else{
    while((readReg(RM3100_STATUS_REG) & 0x80) != 0x80); //read internal status register
  }
  Wire.beginTransmission(RM3100Address);
  Wire.write(0x24); //request from the first measurement results register
  Wire.endTransmission();
  // Request 9 bytes from the measurement results registers
```

```
  Wire.requestFrom(RM3100Address, 9);
if(Wire.available() == 9) {     x2 =
Wire.read();     x1 = Wire.read();     x0 =
Wire.read();       y2 = Wire.read();     y1 =
Wire.read();     y0 = Wire.read();       z2 =
Wire.read();     z1 = Wire.read();     z0 =
Wire.read();
  }
  //special bit manipulation since there is not a 24 bit signed int data type
if (x2 & 0x80){
x = 0xFF;
  }
  if (y2 & 0x80){
y = 0xFF;
  }
  if (z2 & 0x80){
z = 0xFF;
  }
  //format results into single 32 bit signed value
  x = (x * 256 * 256 * 256) | (int32_t)(x2) * 256 * 256 | (uint16_t)(x1) * 256 | x0;
y = (y * 256 * 256 * 256) | (int32_t)(y2) * 256 * 256 | (uint16_t)(y1) * 256 | y0;   z
= (z * 256 * 256 * 256) | (int32_t)(z2) * 256 * 256 | (uint16_t)(z1) * 256 | z0;
  //calculate magnitude, B,  of results from Bx, By and Bz values
  double uT = sqrt(pow(((float)(x)/gain),2) + pow(((float)(y)/gain),2)+ pow(((float)(z)/gain),2));
  //display results
  Serial.print("Data in counts:");
  Serial.print("   X:");
  Serial.print(x);
  Serial.print("   Y:");
  Serial.print(y);
  Serial.print("   Z:");
  Serial.println(z);
  Serial.print("Data in microTesla(uT):");
  Serial.print("   X:");
  Serial.print((float)(x)/gain);
  Serial.print("   Y:");
  Serial.print((float)(y)/gain);
  Serial.print("   Z:");
  Serial.println((float)(z)/gain);
  Serial.print("Magnitude(uT):");
  Serial.println(uT);
  Serial.println();
// End of printout
}
//addr is the 7 bit value of the register's address (without the R/W bit) uint8_t
readReg(uint8_t addr){
```

```
  uint8_t data = 0;
  // Enable transmission to specific which register to read from
  Wire.beginTransmission(RM3100Address);
  Wire.write(addr); //request from the REVID register
Wire.endTransmission();
  delay(100);
  // Request 1 byte from the register specified earlier
Wire.requestFrom(RM3100Address, 1);   if(Wire.available()
== 1) {
    data = Wire.read();
  }
  return data;
}
//addr is the 7 bit (No r/w bit) value of the internal register's address, data is 8 bit data being written void
writeReg(uint8_t addr, uint8_t data){
  Wire.beginTransmission(RM3100Address);
  Wire.write(addr);
  Wire.write(data);
  Wire.endTransmission();
}
//newCC is the new cycle count value (16 bits) to change the data acquisition void
changeCycleCount(uint16_t newCC){
  uint8_t CCMSB = (newCC & 0xFF00) >> 8; //get the most significant byte
uint8_t CCLSB = newCC & 0xFF; //get the least significant byte
  Wire.beginTransmission(RM3100Address);
  Wire.write(RM3100_CCX1_REG);
  Wire.write(CCMSB);  //write new cycle count to ccx1
  Wire.write(CCLSB);  //write new cycle count to ccx0
  Wire.write(CCMSB);  //write new cycle count to ccy1
  Wire.write(CCLSB);  //write new cycle count to ccy0
  Wire.write(CCMSB);  //write new cycle count to ccz1
  Wire.write(CCLSB);  //write new cycle count to ccz0
  Wire.endTransmission();
}
```

Step 16) Open the IDE development window by clicking on the desktop icon.

Step 17) Under ‘File’ click on ‘New’ to open a new blank sketch window.

Step 18) Delete the template code so that only the index for the first line appears.

Step 19) Cut and paste the above sketch code into the window.

Step 20) Under ‘File’ click on ‘Save as’ and save your sketch with a new name into a convenient folder. The IDE program will create a folder in the selected directory with the same name as the sketch. In this folder will be the ‘.INO’ file, which is the sketch code. To

run the sketch from scratch in the future, go to the Arduino folder you created and open it. Click on the .ino file for the sketch. It will automatically open in the IDE platform. Alternatively, click on the IDE icon on your desktop, and under 'Files' click on 'Open'. Then find the folder you created and click on the .ino sketch file.

Step 21) Select the options you want to run in Lines 17, 18 and 19. The default Cycle Count (initialCC) in Line 17 is 200 samples per second or 0xC8 in hexadecimal. In Line 18, set 'singleMode' to "1" for continuous measuring or "0" for only 1 measurement. In Line 19, use the default signaling method using the DRDY pin set to 1. Under' File' click on 'Save' to save your changes.

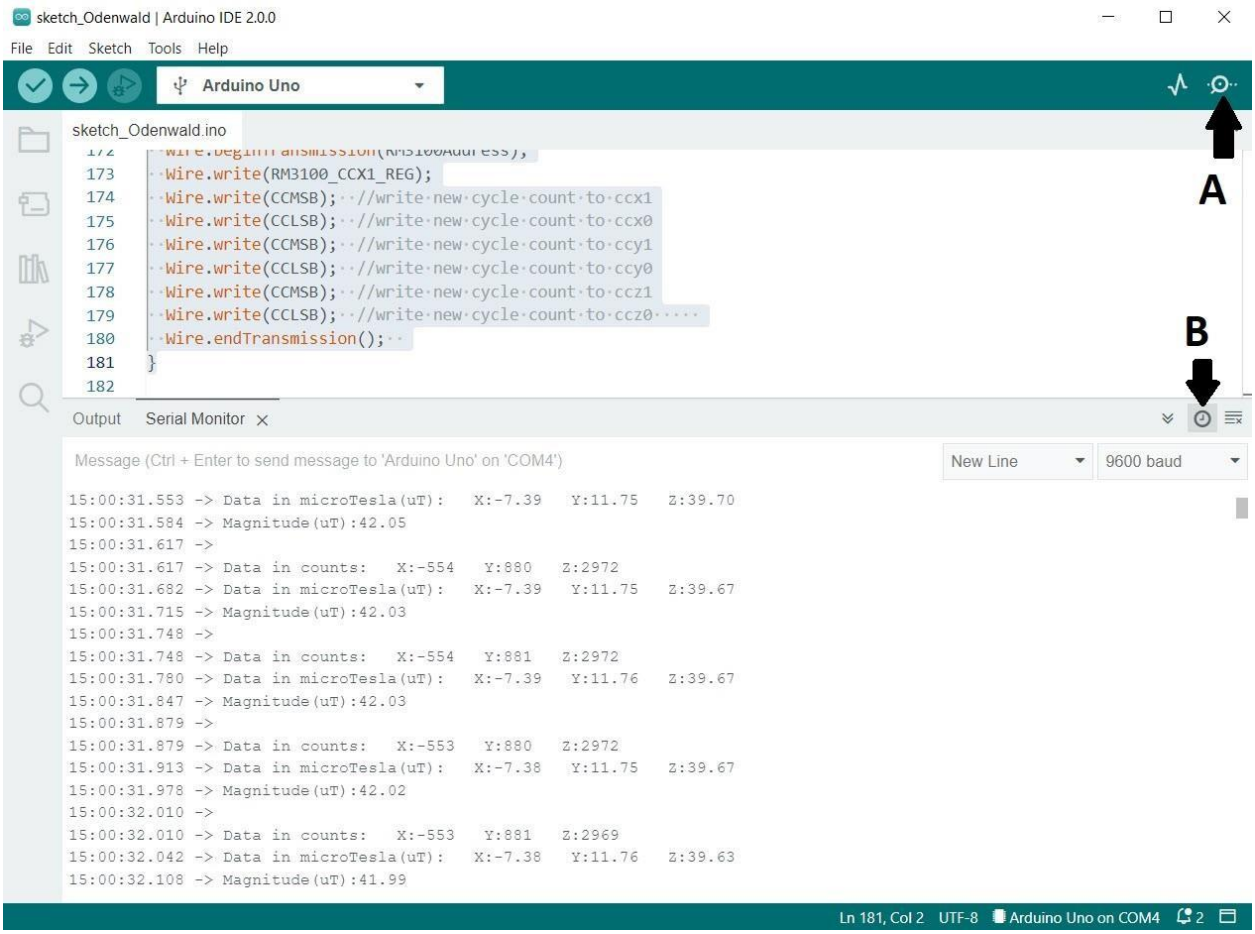


Figure 14. Example of magnetometer output.

Step 22) Plug in the USB cable between the Arduino board and your laptop/computer.

Step 23) With the IDE platform running, check using 'Tools' that the IDE has correctly detected the Arduino board an assigned it a COM channel as you did in Step 6.

Step 24) Once the desired Options are selected in Step 21, press the 'upload' button to load the sketch into the Arduino.

Step 25) Press the Serial button in the top-right of the IDE window (Arrow A) to open the Serial display window. This is where the RM3100 output will be written for you to see. Click on Arrow B and select 'time stamp' to get the hh:mm:ss for each measurement at the start of each row.

What should happen at this point if all the jumper connections are correct and the code compiled without errors is that the sketch will continuously output values as shown in Figure 14. The output in the Serial Monitor window will continue to scroll with new values until the program is stopped. If you are not able to troubleshoot the problem, PNI Corp. has a very good Help Desk that will get a staff engineer to advise you. They can be reached at https://www.pnicorp.com/contact/

To stop the script from executing, you can upload a new 'dummy sketch' such as this one

```
void setup() {
 Serial.begin(9600);
 Serial.println("Stop this stuff!!!!");
}
void loop() {
}
```

Step 26) The manufacturer's output format is not conducive for capturing the data in a form suitable for importing into a spreadsheet. To correct this problem, we must substitute the following code block

```
  //display results
  Serial.print("Data in counts:");
  Serial.print("   X:");
  Serial.print(x);
  Serial.print("   Y:");
  Serial.print(y);
  Serial.print("   Z:");
  Serial.println(z);
  Serial.print("Data in microTesla(uT):");
  Serial.print("   X:");
  Serial.print((float)(x)/gain);
  Serial.print("   Y:");
  Serial.print((float)(y)/gain);
  Serial.print("   Z:");
  Serial.println((float)(z)/gain);
  Serial.print("Magnitude(uT):");
  Serial.println(uT);
  Serial.println();  //
End of printout
```

With the following code block:

```
  //display results
   Serial.print((float)(x)/gain,4);
   Serial.print(" , ");
   Serial.print((float)(y)/gain,4);
   Serial.print(" , ");
   Serial.println((float)(z)/gain,4);
   Serial.print(" , ");
   Serial.print(uT,4);
   Serial.print(" , "); //
End of printout
```

This will produce the output to the screen shown in Figure 15 consisting of Bx, By, Bz, Btotal in μT units. The number '4' in the print statement is the number of decimal places for the number.

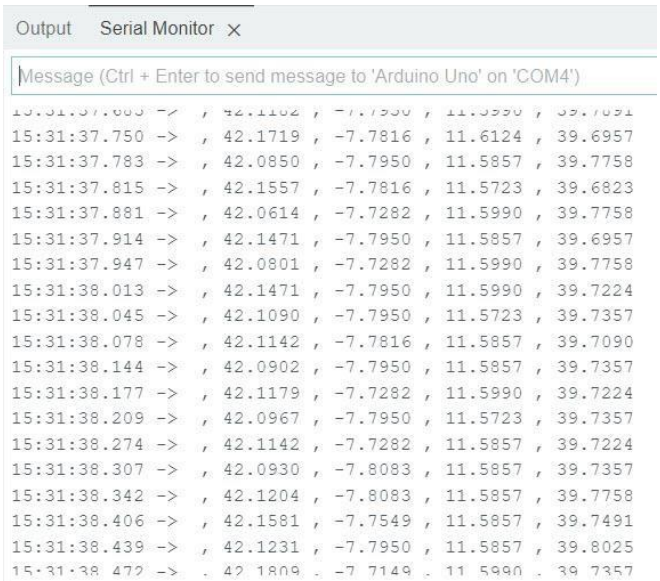


Figure 15. Example of output format with data in four decimal places.

4. Exporting the Data to Excel.

The data generated by the Arduino is presented in real time in the Serial Monitor window but is not captured for future study. Unfortunately, you cannot highlight the numbers in the Serial Monitor window and cut-and-paste them into the spreadsheet. You will have to open an Excel spreadsheet and transfer the numbers by hand into separate rows and columns. One way to do this is to take a picture of the display such as the one in Figure 15 and then import it into the spreadsheet as an image. Then you can easy transfer the numbers

with the image in full view. For example, the screengrab in Figure 15 converted into a spreadsheet – by hand- looks like Figure 16.

A capture of only seven measurements reveals that for this example, the strength of the magnetic field at the test location (desktop) was:

- Bx = 42.12 ± 0.044 μT
- By = -7.78± 0.024 μT
- Bz = 11.59 ±0.013 μT
- B_{total} = 39.73 ± 0.04 μT

It is noteworthy that, without even attempting to make the environment magnetically clean from artificial sources, the 'noise level' (i.e. standard deviation) of the data is about 13 to 44 nanoTeslas. This is already 10 times better than previous magnetometer designs involving suspended mirrors, Hall effect sensors and photoelectric sensors.

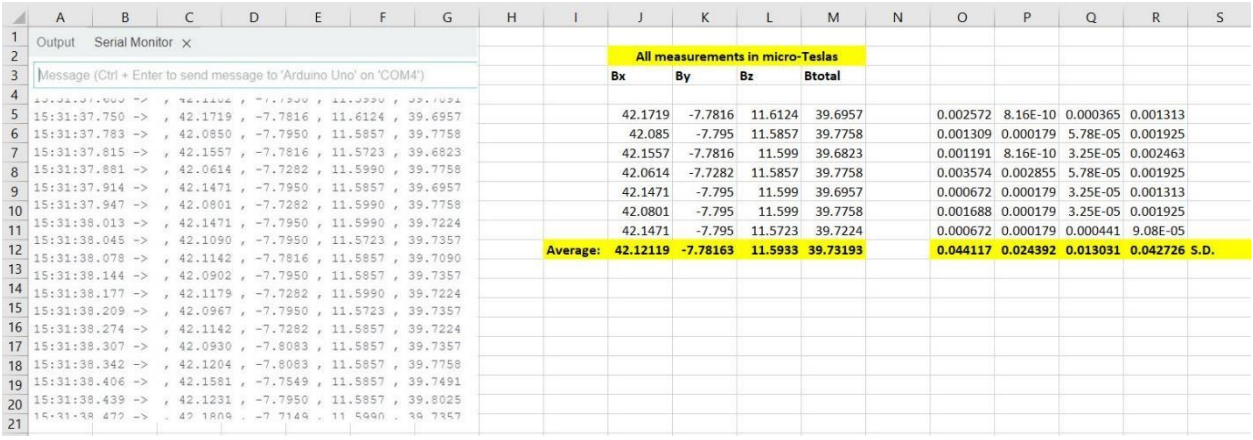


Figure 16. Spreadsheet with imported data.

Taking Measurements

The arrow printed on the RM3100 modules indicates the intended line-of-sight. The RM3100 modules are arranged in a north-east-down (NED) coordinate system, and the arrow is parallel to the x-axis sensor. When the module is pointing directly magnetic south, the x-axis reading will be maximized, and the y-axis will be zero. In similar fashion, when the module is pointing west, the y-axis reading will be maximized, and the x-axis reading will be zero. The z-axis reading will depend on the dip angle at the given location. At the geomagnetic equator, where Earth's magnetic field is horizontal, the z-axis reading will be zero when flat.

The sensors have a specified linear regime of ±200 μT. (Earth's field is ~50 μT.) To ensure the sensors operate in their linear regime, do not place the RM3100 close to large electric

currents, large masses of ferrous material, or devices incorporating permanent magnets, such as speakers and electric motors.

- Place the RM3100 Testing Board away from changing magnetic fields. If this is not possible, but the local magnetic field is known to have multiple states, try to take readings only when the field is in a known state. For instance, if a motor runs part of the time, take readings only when the motor is in a known state.

- If you are uncertain about the effect a specific interference source may have on the system, place the RM3100 Testing Boards on a firm surface and gradually bring the source in question close to the board, then note when the magnetic field starts to change. If the component cannot be moved, then gradually move the RM3100 module toward the component, carefully ensuring that the orientation of the board remains constant while doing this.

- To have optimum resolution, C = 400 or 800 is required: these values have gain 6.7 and 3.3 nT/LSB and take 13 and 26 milliseconds respectively for a measurement (average of C cycles of all three components). C=800 gives a 3nT digital accuracy and averaged over 32 such measurements (total time 0.8 sec). This value is then saved to a laptop[13].

Table 5. How code parameter, C, influences data acquisition

Cycles	C=50	C=100	C=200	C=400	C=800
LSB / mT	20	38	75	150	303
nT / LSB	50	26	13	6.7	3.33
Rate (Hz)	533	283	147	80	40
Noise (nT)	0	20	15	11	8

[13] https://www.liverpool.ac.uk/~cmi/mag/magChip.html

Part III. Design improvement for averaging and logging

1. Background

The previous design was a bare-bones system in which no changes were made to the Arduino code to optimize the data-averaging operation for geomagnetic storm studies. Typically, the measurements were generated at a rate of one measurement of the three components every 0.05 seconds (20 Hz). Geomagnetic storms and other disturbances at this level of change (± 50 nT) typically occur at timescales of minutes to hours. The current improved version averages the data into one-minute intervals and also computes the B, H and D components of the magnetic field where

$$B = \sqrt{Bx^2 + By^2 + Bz^2}$$

$$H = \sqrt{Bx^2 + By^2}$$

$$D = atan\left(\frac{Bx}{By}\right)$$

Note that D is the angle towards Magnetic North and is the deflection angle measured by previous magnetometer designs based on the displacement of a suspended bar magnet.

2. Creating the Arduino code

The Arduino ‘sketch’ that implements the new block-averaging process is shown here, with additional code added in red.

```
/*
////////////////////////////////////////////////////////////////////// //////////////////////////////////////////////////////////////////////////////////

This program uses the RM3100 magnetometer developed by PNI Sensors Corp. The
base code was published by PNI on GitHub.com to interrogate the magnetometer
using a basic Arduino Uno FR3 microcontroller.

The program makes measurements of Bx, By, Bz and calculated B, H and D.
The measurements are averaged into 1-minute values and outputted on the
Serial Monitor, one line per minute in the format Bx, By, Bz, B, H, D with 4-
decimal places accuracy. The output units for Bx, By, Bz, B and H  are in
microTeslas. The units for the displacement angle, D, are in degrees.
```

```
This code was modified by Dr. Sten Odenwald, NASA HEAT Program, Goddard Space
Flight Center on November 1, 2022.
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
*/

#include <Arduino.h>
#include <Wire.h>
#define RM3100Address 0x20 // RM3100 slave address: Pin 2 and 4 set to LOW

//pin definitions
#define PIN_DRDY 8 //Set pin D8 to be the Data Ready Pin

//internal register values without the R/W bit
#define RM3100_REVID_REG 0x36 // Hexadecimal address: Revid internal register
#define RM3100_POLL_REG 0x00 // Hexadecimal address:Poll internal register
#define RM3100_CMM_REG 0x01 // Hexadecimal address: CMM internal register
#define RM3100_STATUS_REG 0x34 //Hexadecimal address:Status internal register
#define RM3100_CCX1_REG 0x04 // Hexadecimal address: Cycle Count X1 register #define
RM3100_CCX0_REG 0x05 // Hexadecimal address: Cycle Count X0 register
/*
Cycles         C=50   C=100   C=200   C=400   C=800
LSB / mT        20    38      75      150     303 nT / LSB
50    26      13      6.7     3.33 Rate (Hz)      533    283
147     80      40
Noise (nT)      0     20      15      11      8

This code is designed for C=800 which provides an expected  data
noise near 3.3 nanoTeslas.
*/

//options
#define initialCC 800 // Set the cycle count to 800 for 3.3 nT accuracy
#define singleMode 0 //0 =continuous measurement; 1 = single measurement
#define useDRDYPin 1 //0 = not using DRDYPin ; 1 = using DRDYPin to for data
#define NO_TO_AVE     2158      // Number of measurements to average

 int Nmaximum = NO_TO_AVE;

// for C = 800
// selecting NO_TO_AVE adds a second level of block averaging to the C=800
// sample average. Example with C=800,  Nmaximum = 64 provides an
// effective N = 800x64 = 51,200-measurement averaging to get to the
```

```
// longer time scales.

//Nmaximum = 64 .... 1.8 second averaging
//Nmaximum = 100 .... 2.8 second averaging
//Nmaximum = 1000 ... 28.0 second averaging
//Nmaximum = 2000 ... 56.6 second averaging
//Nmaximum = 2158 ... 60.0 second averaging

uint8_t revid;
uint16_t cycleCount;
float gain; float Bx,
By, Bz;
float B; float
H; float D;
float cscale; // the factor to multiply each summed components to get the average int
count; // this is the counter for the number of measurements in one loop

void setup() {
  pinMode(PIN_DRDY, INPUT);
pinMode(SCL,INPUT_PULLUP);  //enable SCL pull-up resistor
pinMode(SDA,INPUT_PULLUP);  //enable SDA pull-up resistor
  Wire.begin(); // Initiate the Wire library
Serial.begin(9600); //set baud rate to 9600
  delay(500); // set delay to 0.5 seconds or 500 milliSeconds between measurements

  revid = readReg(RM3100_REVID_REG);
  Serial.print("REVID ID = 0x"); //REVID ID should be 0x22
  Serial.print(revid, HEX);

  changeCycleCount(initialCC); //change the cycle count; default = 200

  cycleCount = readReg(RM3100_CCX1_REG);
  cycleCount = (cycleCount << 8) | readReg(RM3100_CCX0_REG);

  Serial.print("      Cycle Counts = "); //display cycle count
  Serial.print(cycleCount);

  //linear equation to calculate the gain from cycle count
gain = (0.3671 * (float)cycleCount) + 1.5;

  Serial.print("      Gain = "); //display gain; default gain should be around 75 for the default cycle count of
200
  Serial.println(gain);

  if (singleMode){
```

```
    //set up single measurement mode
  writeReg(RM3100_CMM_REG, 0);
    writeReg(RM3100_POLL_REG, 0x70);
  }
else{
   // Enable transmission to take continuous measurement with Alarm functions off
writeReg(RM3100_CMM_REG, 0x79);
  }
}

float totalB, totalBx, totalBy, totalBz, totalH, totalD;

void loop() {
long x = 0;   long
y = 0;   long z =
0;

  uint8_t x2,x1,x0,y2,y1,y0,z2,z1,z0;
double BMagnitude;                          // define the dummy variable for the field strength magnitude
cscale = 1.0/(float)Nmaximum;              // the factor to multiply each summed components to get the
average

  //wait until data is ready using 1 of two methods (chosen in Options)
if(useDRDYPin){
   while(digitalRead(PIN_DRDY) == LOW); //check RDRY pin
  }
  else{
   while((readReg(RM3100_STATUS_REG) & 0x80) != 0x80); //read internal status register
  }

  Wire.beginTransmission(RM3100Address);
  Wire.write(0x24); //request from the first measurement results register
  Wire.endTransmission();

  // Request 9 bytes from the measurement results registers
Wire.requestFrom(RM3100Address, 9);   if(Wire.available()
== 9) {     x2 = Wire.read();     x1 = Wire.read();     x0 =
Wire.read();

   y2 = Wire.read();
y1 = Wire.read();     y0
= Wire.read();   z2 =
Wire.read();   z1 =
Wire.read();   z0 =
Wire.read();
  }
```

```
  //special bit manipulation since there is not a 24 bit signed int data type
if (x2 & 0x80){
x = 0xFF;
  }
  if (y2 & 0x80){
y = 0xFF;
  }
  if (z2 & 0x80){
z = 0xFF;
  }
  //format results into single 32 bit signed value
  x = (x * 256 * 256 * 256) | (int32_t)(x2) * 256 * 256 | (uint16_t)(x1) * 256 | x0;
y = (y * 256 * 256 * 256) | (int32_t)(y2) * 256 * 256 | (uint16_t)(y1) * 256 | y0;   z
= (z * 256 * 256 * 256) | (int32_t)(z2) * 256 * 256 | (uint16_t)(z1) * 256 | z0;

  //calculate magnitude, B, and H of results from Bx, By and Bz values
      BMagnitude = sqrt(pow(((float)(x)/gain),2) + pow(((float)(y)/gain),2)+ pow(((float)(z)/gain),2));
     B = BMagnitude;  //  this is the total field strength in nanoTeslas
     BMagnitude = sqrt(pow(((float)(x)/gain),2) + pow(((float)(y)/gain),2));
     H = BMagnitude; // this is the magnitude of the horizontal plane x-y projection of B vector


  //display results
  Bx=(float)(x)/gain;
  By = (float)(y)/gain;
  Bz= (float)(z)/gain;
  // now calculate the displacement angle
  D = atan(Bx/By)*57.296; // this is the angle in degree units

// add the measurement to each summing array for averaging
totalBx += Bx*cscale; totalBy += By*cscale; totalBz +=
Bz*cscale; totalB += B*cscale; totalH += H*cscale; totalD +=
D*cscale;
/*
// This block of print statements is useful for diagnosing if the average is  //
being computed correctly.
   Serial.print(" ...count= ");
   Serial.print(count);
   Serial.print(" , ");
   // Serial.print("   X:");
   Serial.print(Bx,4);   // print Bx to 4-decimal points
   Serial.print(" , ");
   // Serial.print("   Y:");
   Serial.print(By,4);
   Serial.print(" , ");
```

```
  // Serial.print("   Z:");
  Serial.print(Bz,4);
  Serial.print(" , ");
  Serial.print(B,4);  //print the magnitude of B
  Serial.print(" , ");
  Serial.print(H,4);  //print projection of B on horizontal plane
  Serial.print(" , ");
  Serial.println(D,4); //print the deviation angle in degrees then skip a line
*/
count++;
if(count==Nmaximum)
{
  // finished averaging NO_TO_AVE measurements in this loop now print out and reset count index //
Serial.print(" The average values are  ");
  Serial.print(totalBx,4);
  Serial.print(" , ");
  // Serial.print("   Y:");
  Serial.print(totalBy,4);
  Serial.print(" , ");
  // Serial.print("   Z:");
  Serial.print(totalBz,4);
  Serial.print(" , ");
  Serial.print(totalB,4);  //print the magnitude of B
  Serial.print(" , ");
  Serial.print(totalH,4);  //print projection of B on horizontal plane
  Serial.print(" , ");
  Serial.println(totalD,4); //print the deviation angle in degrees then skip a line
// initialize summ variables
totalBx=0;// initialize the sum for Bx to 0.0
totalBy=0;// initialize the sum for By to 0.0
totalBz=0;// initialize the sum for Bz to 0.0
totalB=0;   // initialize the sum for B to 0.0
totalH=0;// initialize the sum for H to 0.0 totalD=0;//
initialize the sum for D to 0.0
count=0;  // reset count to zero for next block  ---- DOESNT WORK
}
//End of added code for averaging and output
}// end of Void Loop()
//addr is the 7 bit value of the register's address (without the R/W bit) uint8_t
readReg(uint8_t addr){
  uint8_t data = 0;

  // Enable transmission to specific which register to read from
  Wire.beginTransmission(RM3100Address);
  Wire.write(addr); //request from the REVID register
  Wire.endTransmission();
```

```
  delay(100);

  // Request 1 byte from the register specified earlier
Wire.requestFrom(RM3100Address, 1);   if(Wire.available()
== 1) {
    data = Wire.read();
  }
  return data;
}

//addr is the 7 bit (No r/w bit) value of the internal register's address, data is 8 bit data being written void
writeReg(uint8_t addr, uint8_t data){
  Wire.beginTransmission(RM3100Address);
  Wire.write(addr);
  Wire.write(data);
  Wire.endTransmission();
}

//newCC is the new cycle count value (16 bits) to change the data acquisition void
changeCycleCount(uint16_t newCC){
  uint8_t CCMSB = (newCC & 0xFF00) >> 8; //get the most significant byte
uint8_t CCLSB = newCC & 0xFF; //get the least significant byte

  Wire.beginTransmission(RM3100Address);
  Wire.write(RM3100_CCX1_REG);
  Wire.write(CCMSB);  //write new cycle count to ccx1
  Wire.write(CCLSB);  //write new cycle count to ccx0
  Wire.write(CCMSB);  //write new cycle count to ccy1
  Wire.write(CCLSB);  //write new cycle count to ccy0
  Wire.write(CCMSB);  //write new cycle count to ccz1
  Wire.write(CCLSB);  //write new cycle count to ccz0
  Wire.endTransmission();
}
```

3. Taking and Analyzing Data

An example of ten minutes of data-taking is shown in Figure 17.

```
51  #define NO_TO_AVE     2158      // Number of measurements to average; p
52  int Nmaximum = 2158;
53
54  uint8_t revid;
55  uint16_t cycleCount;
56  float gain;
```

Output Serial Monitor ×

Message (Ctrl + Enter to send message to 'Arduino Uno' on 'COM4')

```
08:55:50.333 -> REVID ID = 0x22      Cycle Counts = 800      Gain = 295.18
08:56:50.485 -> -6.5914 , -2.3668 , 41.8616 , 42.4435 , 7.0035 , 19.7523
08:57:50.503 -> -6.5895 , -2.3709 , 41.8600 , 42.4418 , 7.0031 , 19.7899
08:58:50.490 -> -6.5794 , -2.3804 , 41.8602 , 42.4409 , 6.9968 , 19.8902
08:59:50.474 -> -6.5805 , -2.3883 , 41.8620 , 42.4433 , 7.0005 , 19.9489
09:00:50.498 -> -6.5760 , -2.3898 , 41.8631 , 42.4438 , 6.9968 , 19.9722
09:01:50.476 -> -6.5760 , -2.3888 , 41.8556 , 42.4363 , 6.9964 , 19.9644
09:02:50.538 -> -6.5692 , -2.3924 , 41.8625 , 42.4424 , 6.9913 , 20.0132
09:03:50.540 -> -6.5624 , -2.3918 , 41.8540 , 42.4327 , 6.9847 , 20.0258
09:04:50.535 -> -6.5510 , -2.4013 , 41.8555 , 42.4331 , 6.9773 , 20.1321
09:05:50.571 -> -6.5429 , -2.4015 , 41.8494 , 42.4258 , 6.9697 , 20.1561
```

Figure 17. Example of output in 10-minute cadence mode. The data are Bx, By, Bz, B, H, D.

By importing this data into an Excel spreadsheet, by hand, the prolonged averaging in one-minute data blocks yields the spreadsheet shown in Figure 18.

```
#define NO_TO_AVE     2158      // Number of measurements to average; p
int Nmaximum = 2158;

uint8_t revid;
uint16_t cycleCount;
float gain;
```

Output Serial Monitor ×

Message (Ctrl + Enter to send message to 'Arduino Uno' on 'COM4')

```
08:55:50.333 -> REVID ID = 0x22      Cycle Counts = 800      Gain = 295.18
08:56:50.485 -> -6.5914 , -2.3668 , 41.8616 , 42.4435 , 7.0035 , 19.7523
08:57:50.503 -> -6.5895 , -2.3709 , 41.8600 , 42.4418 , 7.0031 , 19.7899
08:58:50.490 -> -6.5794 , -2.3804 , 41.8602 , 42.4409 , 6.9968 , 19.8902
08:59:50.474 -> -6.5805 , -2.3883 , 41.8620 , 42.4433 , 7.0005 , 19.9489
09:00:50.498 -> -6.5760 , -2.3898 , 41.8631 , 42.4438 , 6.9968 , 19.9722
09:01:50.476 -> -6.5760 , -2.3888 , 41.8556 , 42.4363 , 6.9964 , 19.9644
09:02:50.538 -> -6.5692 , -2.3924 , 41.8625 , 42.4424 , 6.9913 , 20.0132
09:03:50.540 -> -6.5624 , -2.3918 , 41.8540 , 42.4327 , 6.9847 , 20.0258
09:04:50.535 -> -6.5510 , -2.4013 , 41.8555 , 42.4331 , 6.9773 , 20.1321
09:05:50.571 -> -6.5429 , -2.4015 , 41.8494 , 42.4258 , 6.9697 , 20.1561
```

Sample	Bx	By	Bz	B	H	D
1	-6.5914	-2.3668	41.8616	42.4435	7.0035	19.7523
2	-6.5895	-2.3709	41.86	42.4418	7.0031	19.7899
3	-6.5794	-2.3804	41.8602	42.4409	6.9968	19.8902
4	-6.5805	-2.3883	41.862	42.4433	7.0005	19.9489
5	-6.576	-2.3898	41.8631	42.4438	6.9968	19.9722
6	-6.576	-2.3888	41.8556	42.4363	6.9964	19.9644
7	-6.5692	-2.3924	41.8625	42.4424	6.9913	20.0132
8	-6.5624	-2.3918	41.854	42.4327	6.9847	20.0258
9	-6.551	-2.4013	41.8555	42.4331	6.9773	20.1321
10	-6.5429	-2.4015	41.8494	42.4258	6.9697	20.1561
Average	**-6.5718**	**-2.3872**	**41.8584**	**42.4384**	**6.99201**	**19.9645**
	µT	**µT**	**µT**	**µT**	**µT**	**degrees**
Sample						
1	0.000383	0.000416	1.03E-05	2.64E-05	0.000132	0.045033
2	0.000312	0.000266	2.59E-06	1.18E-05	0.000123	0.030489
3	5.73E-05	4.62E-05	3.28E-06	6.45E-06	2.29E-05	0.005522
4	7.52E-05	1.21E-06	1.3E-05	2.44E-05	7.21E-05	0.000244
5	1.74E-05	6.76E-06	2.22E-05	2.96E-05	2.29E-05	5.91E-05
6	1.74E-05	2.56E-06	7.78E-06	4.24E-06	1.93E-05	1.21E-08
7	6.92E-06	2.7E-05	1.69E-05	1.63E-05	5.04E-07	0.002371
8	8.89E-05	2.12E-05	1.93E-05	3.2E-05	5.34E-05	0.003756
9	0.000434	0.000199	8.35E-06	2.77E-05	0.000216	0.028086
10	0.000837	0.000204	8.08E-05	0.000158	0.000498	0.036707
± noise	**15.74**	**11.50**	**4.53**	**6.12**	**11.35**	**0.13**
	nanoT	**nanoT**	**nanoT**	**nanoT**	**nanoT**	**degrees**

Figure 18. Example of spreadsheet for 1-minute cadence data.

The results of the averaging and s.d. calculations yield for a 10-minute, 10-sample test session

Bx = -6.572 ± 0.0157 µT
By = -2.387 ± 0.0115 µT
Bz = 41.858 ± 0.0045 µT
B = 42.438 ± 0.0061 µT
H = 6.992 ± 0.0114 µT
D = 19.965 ± 0.13 degrees

The data was not inspected for systematic trends so that the computed s.d representing the measurement noise are upper limits to the random error. Nevertheless, the measurement noise for the magnetic components varies from ±6 to ±16 nT and the deviation angle error is ±0.13° even under these non-optimal measurement and environmental conditions at the 'desktop' and near a laptop. Significant improvements can be made by controlling for environmental noise, systematic trends in a longer sequence of data, and temperature variations.

4. Troubleshooting:

- This is a very sensitive sensor and there can be many problems with local magnetic interference. For example, some of the pin heads on the wires for joining the Arduino and RM3100 boards may be magnetic. Using the 3-inch jumper standoffs will likely reduce this problem.

- Any movement of the RM3100 between measurements will distort the data.
- The Arduino and code do not read out the Bx, By and Bz channels simultaneously but sequentially. This means that if the ambient field changes by a few nT during the millisecond download, the data will be corrupted. A solution is to increase the averaging time between samples so that it is much longer than the data sampling rate.
- Coding errors are the most common problems so the code must be debugged in a logical, step by step manner to identify the errors. It is best to get small parts of the code working before adding new features. Use 'print statements' liberally!
- Remember that the *atan* function returns the angle in radians so you need to convert this to degrees by multiplying by $180/\pi = 57.2958$.

Part IV. Automatic data logging for the Arduino magnetometer

1. Background

If we are only preparing to take a handful of measurements during a geomagnetic storm, the previous, manual method of recording the data would probably work, but a much better solution is to employ an 'automatic' method that creates a proper .csv data file for import into the spreadsheet software. Unfortunately, these methods are a bit more complicated to implement and require a third-party program to capture the data on the COM4 port. The simplest one of these methods uses a program called PuTTY.

PuTTY is a communications tool for running interactive command-line sessions on other computers, usually via the SSH protocol. It can also communicate over a serial port or speak various legacy Internet protocols such as Telnet. PuTTY is a terminal emulator. Back in the day, programmers entered and read information out of mainframe computers with terminals. Some of these were even mechanical and were called teletypes (short for 'Telegraph typewriter'). PuTTY has many available connection types and data transfer protocols, but we are going to use the simple RS232 as serial. Terminal emulators were widely used to connect to other distant computers before the Internet.

2. Procedure

Step 1) PuTTY can be downloaded from. Figure 19 shows the download page. https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

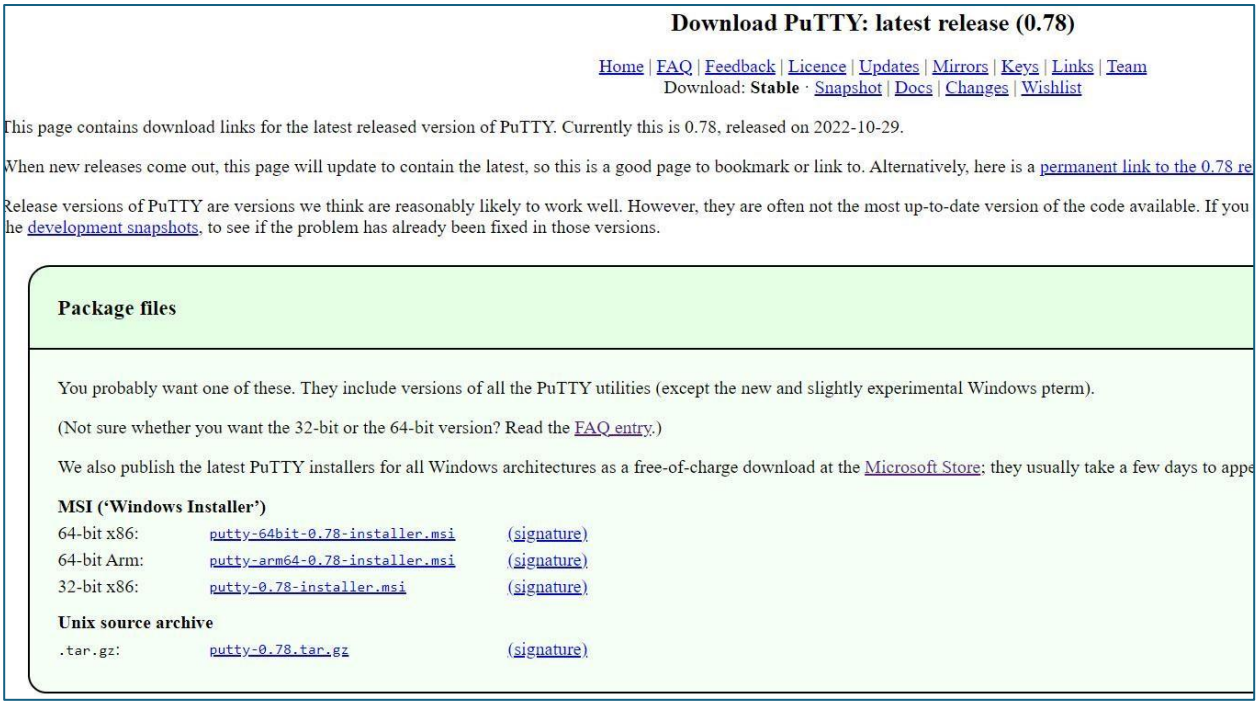


Figure 19. Download page for PuTTY.

Look for the 32- or 64-bit Windows versions. If you're not sure which to download, the 32-bit version is generally the safe option. It will run perfectly well on all processors and on all versions of Windows that PuTTY supports. PuTTY doesn't require to run as a 64-bit application to work well, and having a 32-bit PuTTY on a 64-bit system isn't likely to cause you any trouble.

The 64-bit version will only run if you have a 64-bit processor and a 64-bit edition of Windows (both of these are likely to be true of any recent Windows PC). It will run somewhat faster (in particular, the cryptography will be faster, especially during link setup), but it will consume slightly more memory.

Step 2) Once you have PuTTY downloaded, just run the installer and choose where you want it installed. When installation is completed, open your Systems folder by clicking on the Windows icon on the lower left of your desktop tray. Search for 'PuTTY', open the folder, and drag-and-drop the PuTTY icon (Figure 20) to your desktop.

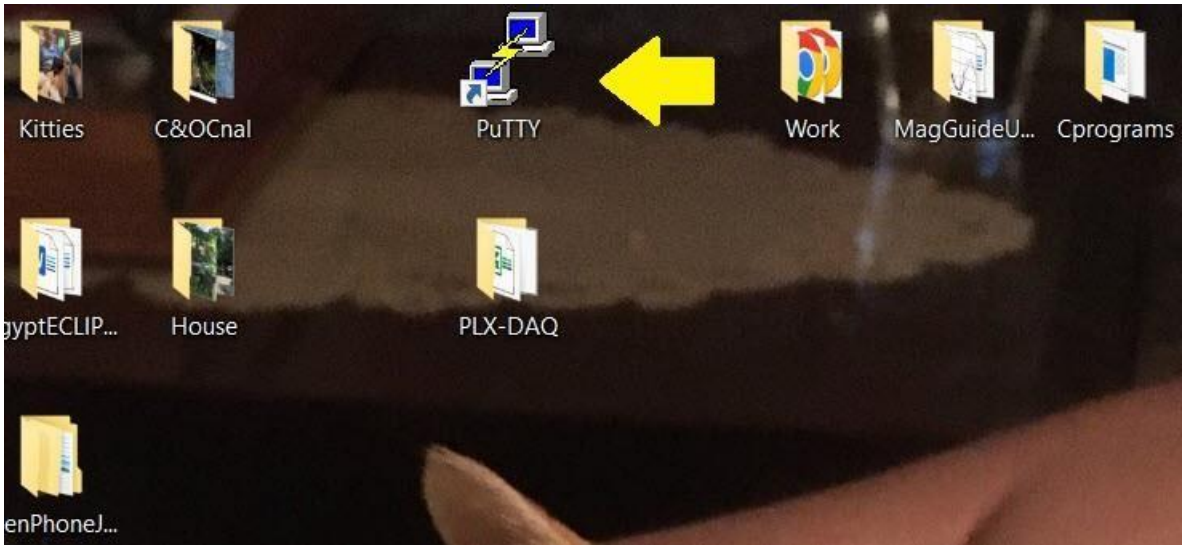


Figure 20. PuTTY icon on desktop window

Step 3) Open the Arduino IDE interface and start the magnetometer program. Make sure the Serial Monitor screen is showing data being gathered and displayed as in Figure 21.

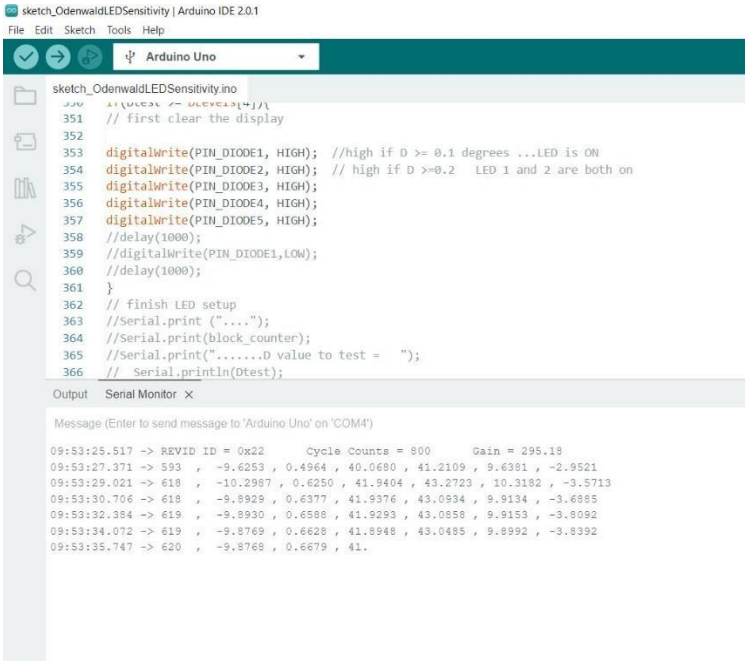


Figure 21. Serial Monitor data display (lower panel) from the magnetometer.

Step 4) Close the Arduino IDE program.

Step 5) Click on the Putty icon to start the program. Enter the following information into the specified windows.

Session:

Serial Line.... COM4
Speed.....9600
Connection Type..... serial ‘ Telnet’

Logging:

Session logging.... printable output
Log file name ..... use ‘Browse’ to select folder and enter file name for output data eg **Data.csv**
What to do if the log file already exists..... Ask the user every time (or your preference)

Connection – serial:

Select a serial line..... COM4
Speed..... 9600
Data bits.....8
Stop bits..... 1
Parity..... None
Flow control..... XON/XOFF

Step 6) Now click ‘open’ and the PuTTY window should immediately open. An example is shown in Figure 22. You should also see the output file (e.g. Data.csv) appear in the target folder.

Step 7) When your data-taking session is finished, click on the ‘x’ on the top-right of the PuTTY window. It will ask you “Are you sure you want to close this session?” click on the ‘OK’ button.

Step 8) Click on your data file to open it using ‘Notepad’ or another pure-text editor.

Step 9) Use your mouse to left click and copy the rows you want to export to Excel.

Step 10) Open Excel and paste the data into a new worksheet. Select ‘Data’ and ‘Text to columns’ and the parameters ‘Delimited’ and ‘comma’ to extract the data into individual columns in the spreadsheet as shown in Figure 23.

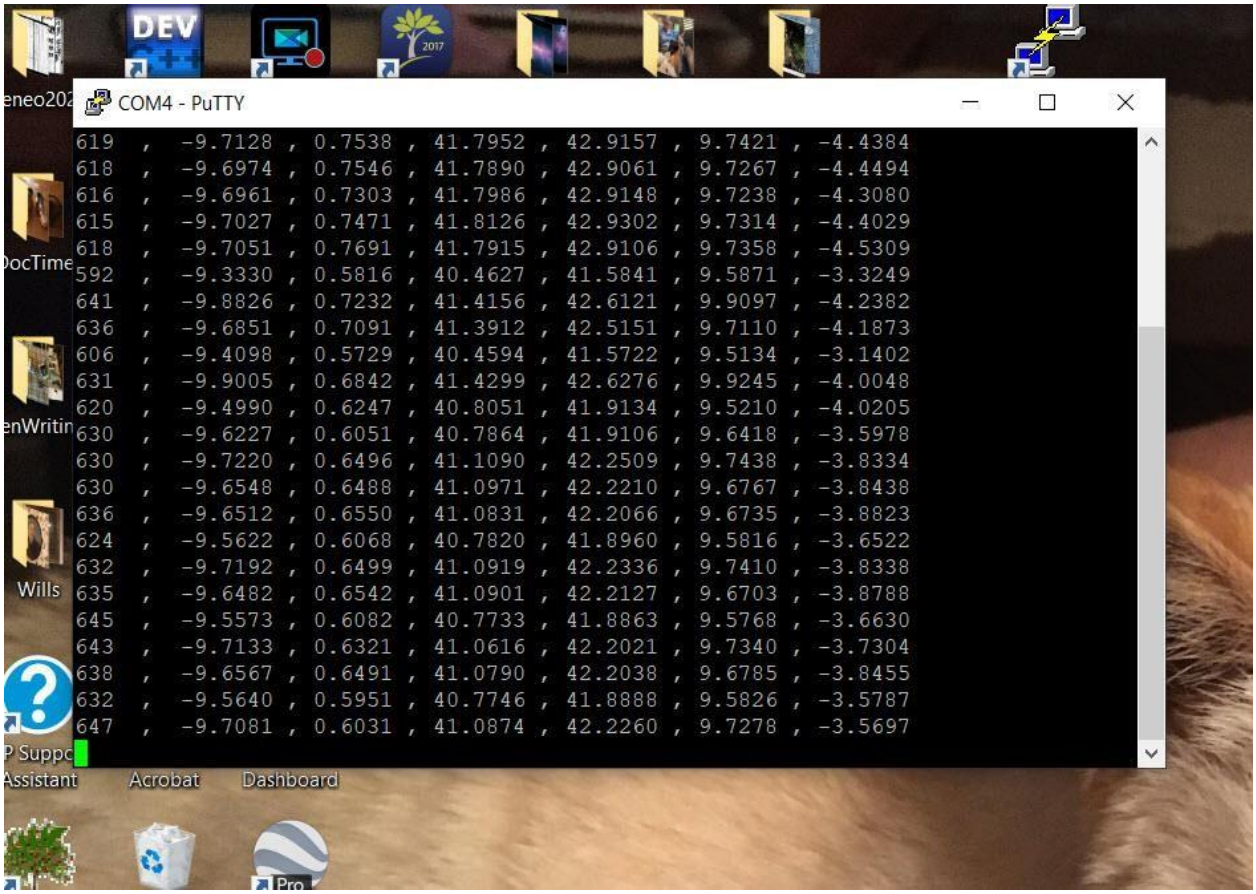


Figure 22. PuTTY window with updating data appearing on desktop.

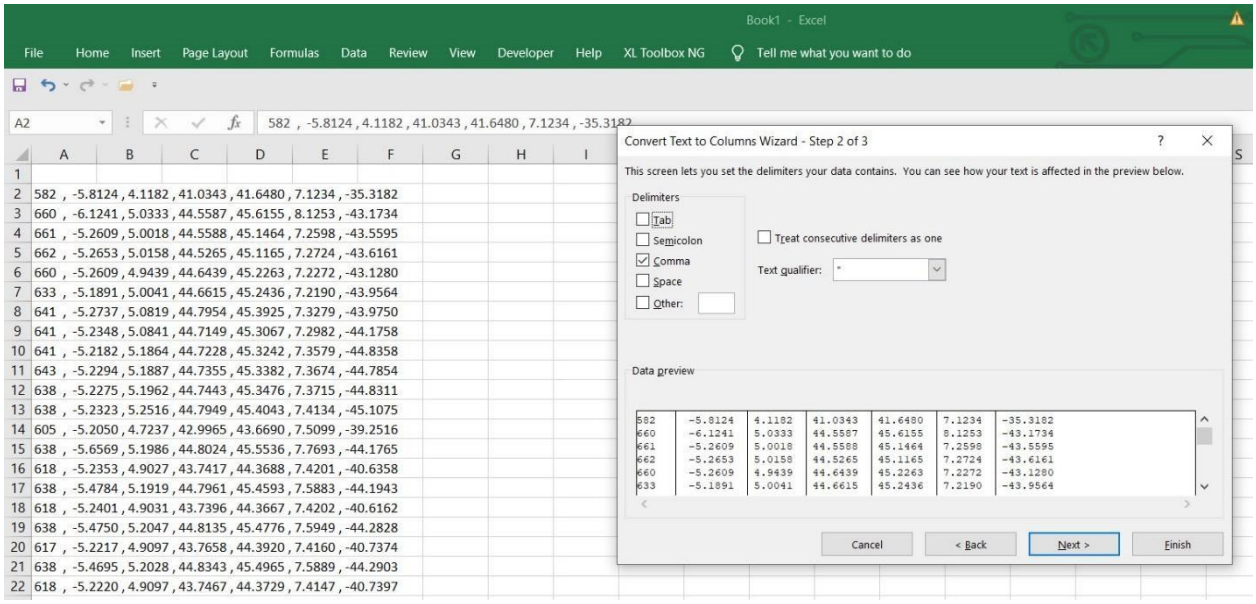


Figure 23. Example of text to columns screen.

Step 11) Click on the 'Next' and 'Finish' buttons and the procedure will insert the data into individual columns as shown in Figure 24. You can now graph the data and perform analysis. An example is shown for about 15 minutes of data. Figure 25 shows 15 minutes of data for the H magnetic component. The H average measurement for N=950 samples correspond to 7.346 ±0.0034 μT. The rms noise is therefore σ = ±3.4 nT. The corresponding value for D is -42.797° ±0.05°. Note that for D there is a large sinusoidal 'trend' in the data so that the computed rms of σ = ±0.05° is an upper limit to the actual noise in the angular measurement. A more careful and controlled environment can reduce these systematic effects significantly.

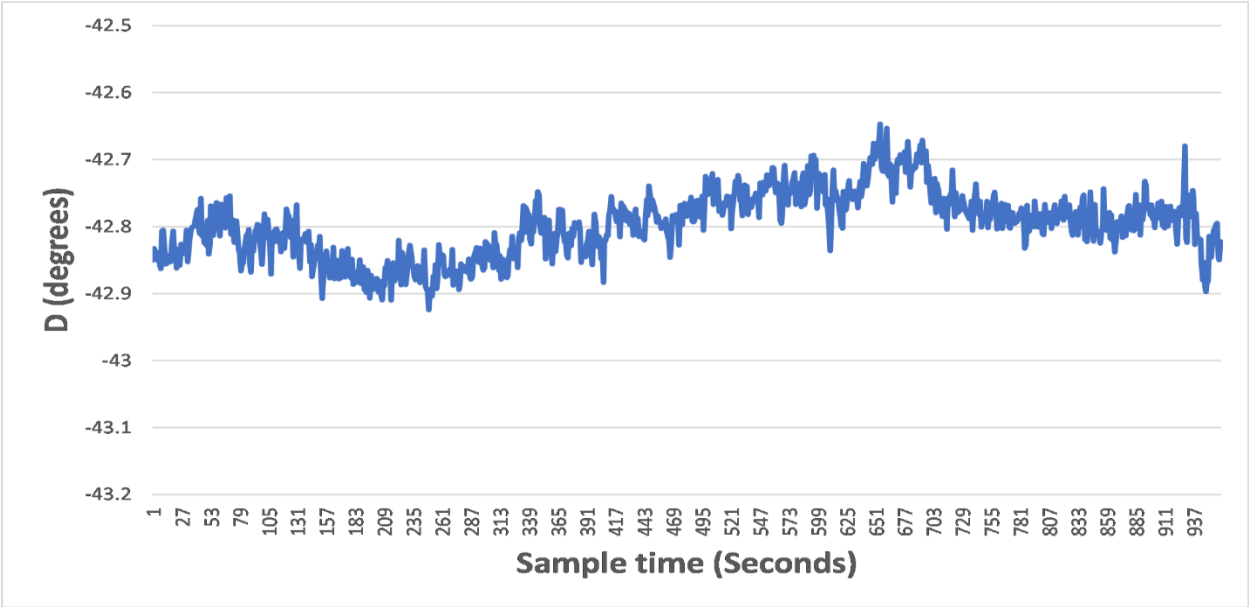


Figure 24. Example of long-term measurement of D component.

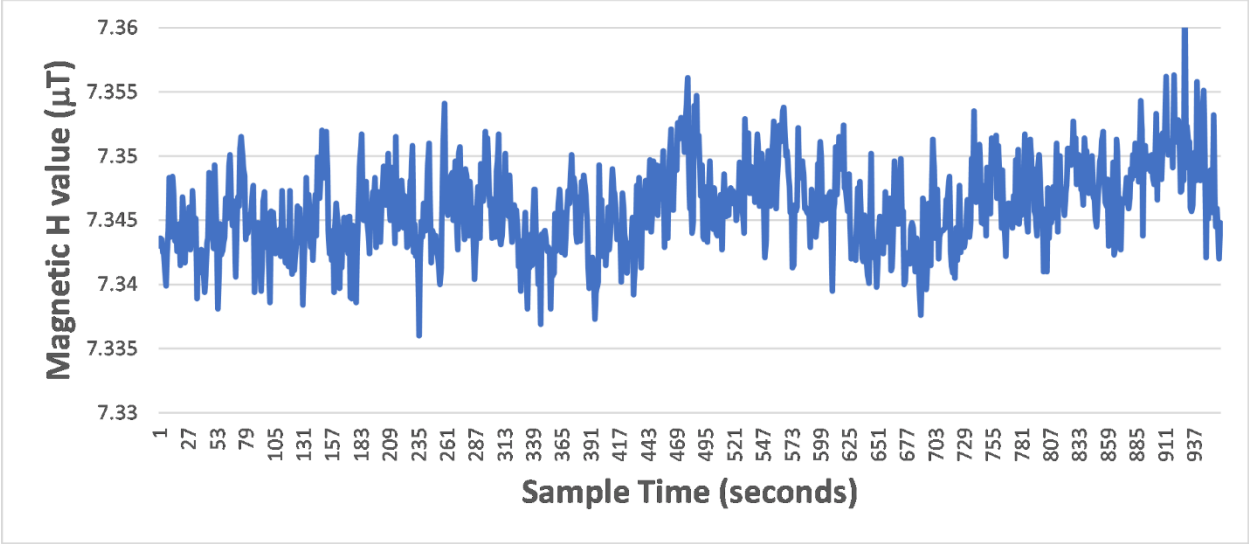


Figure 25. Example of long-term measurement of H magnetic component.

Part V. Detecting strong geomagnetic storms

1. Background:

On April 21, 2023, an Earth-facing filament near Active Region 3283 erupted at 18:12 UTC producing an M1.7 solar flare and a strong coronal mass ejection (CME) shown in Figure 26. The CME arrived at Earth on April 23 at 17:37 UT (1:37 PM EST). It registered as a Kp = 8 (NOAA G4) event and produced aurora seen as far south as the US-Mexican border[14] at +25.9° N latitude and Southern California at +32.5° N. Auroras over Sturgis, South Dakota (+44.4° N), were so big and intense, they surrounded onlookers in all directions.

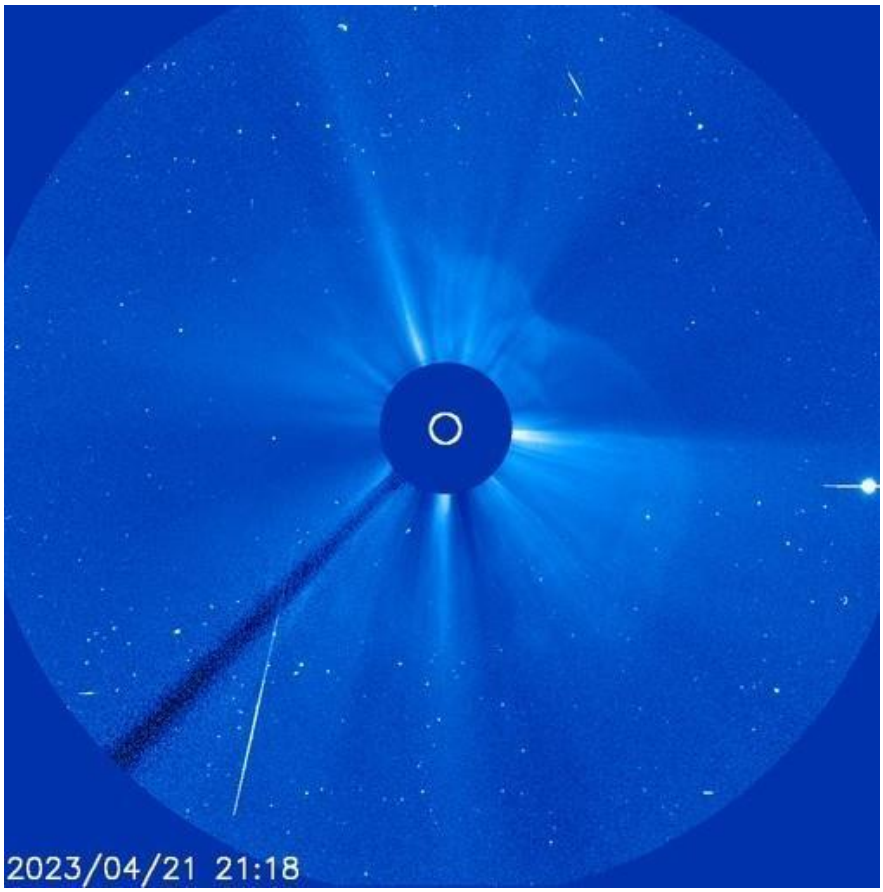


Figure 26. The faint circum-solar 'Halo' CME imaged by SOHO spacecraft on April 21, 2023, at 21:18 UT.

[14] https://spaceweather.com/archive.php?view=1&day=24&month=04&year=2023

A stack plot of magnetic observatory data is shown in Figure 27 from a range of geographic latitudes.

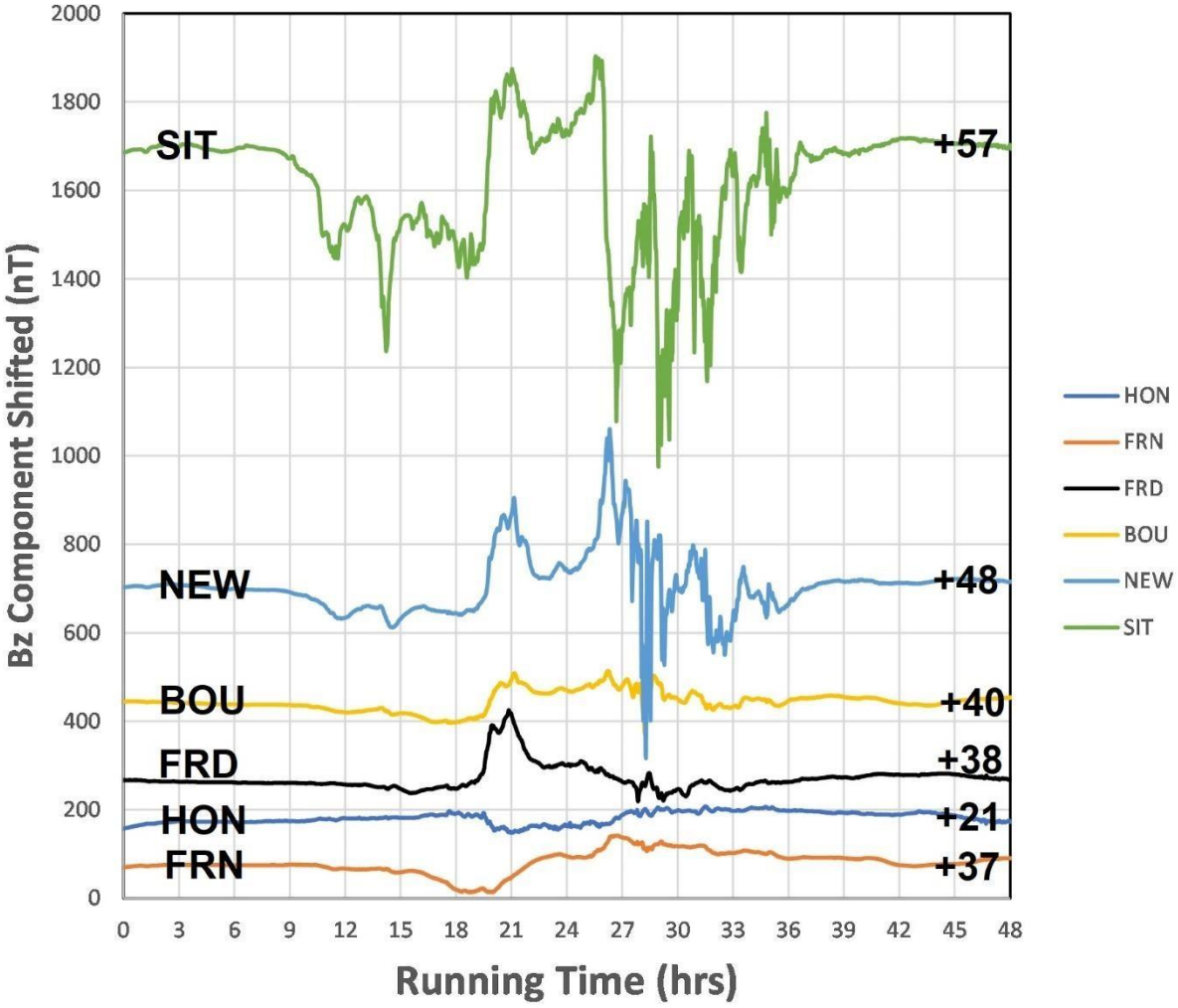


Figure 27. A stack plot of Bz data from a variety of magnetic observatories. It begins at 0:00 UT Midnight on April 23 and continues for 48 hours until Midnight on April 25. The scans are from Fresno (FRN +37N), Honolulu (HON +21N), Fredericksburg (FRD +38N), Boulder (BOU +40N), Newport (NEW +48N) and Sitka (SIT +57N).

Figure 27 shows a clear increase in the geomagnetic variation from low latitudes (FRN and HON) to near-Arctic latitudes (SIT). Although an arbitrary offset in Bz was added to each scan in or der to place it on the figure, the vertical axis shows the difference in Bz between the baseline level for each observatory and the deviation produced by the storm. For example, the Fredericksburg (FRD) variation is about 162 nT while for the Sitka (SIT) event the deviation measured +190 nT to –719 nT for a range of over 900 nT. Clearly, the closer you are to Arctic 'auroral' latitudes, the stronger will be the magnetometer signal.

2. Analysis

There were three sessions of data-taking between April 23 (0:00 UT) and April 25 (0:00 UT) which were edited together to make a continuous 48-hour data series with sampling every minute. The raw ΔBz data is shown in Figure 28.

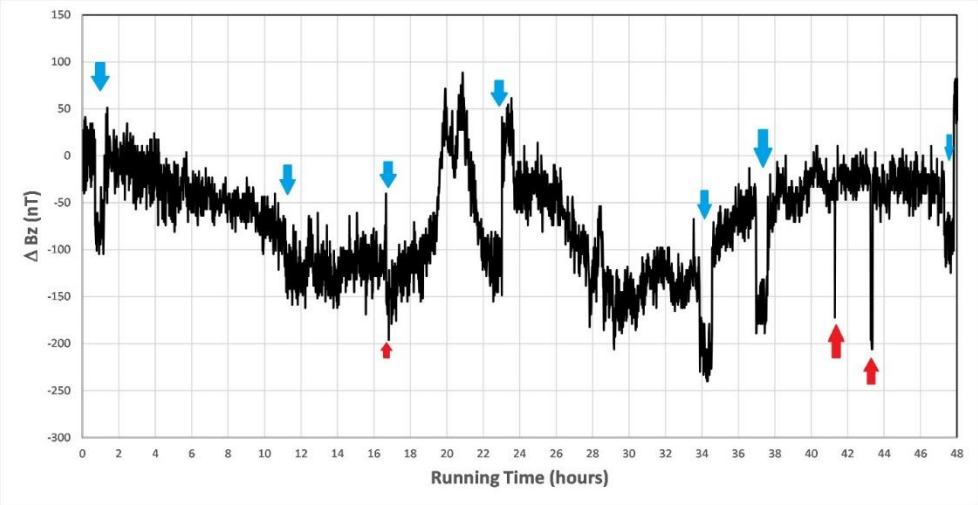


Figure 28. Concatenated data for the Bz data. A constant value of 39.45 μT has been subtracted from each data value so that the residuals for ΔBz can be expressed in units of nT.

The raw data shown in Figure 28 has several jump discontinuities shown with blue arrows and single-data glitches shown in red arrows. These can be removed by simply adding the appropriate 'DC' offset to the subsequent data values and replacing the glitch values by local mean values. The result is the cleaned data shown in Figure 29.

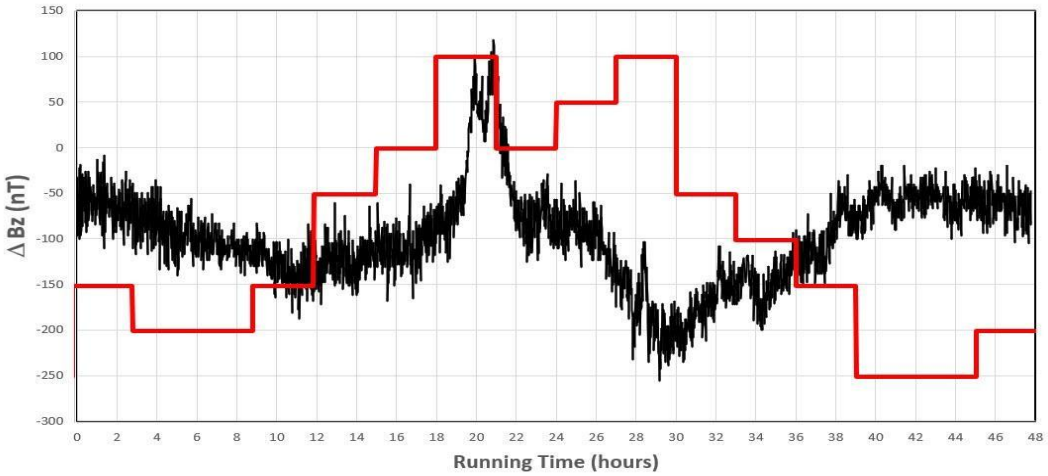


Figure 29. Shifted and de-glitched data for ΔBz. The red lines indicate the corresponding Kp values in 1step increments from a peak value of 8 near times 20 and 28 to a minimum of 1 between times 40 – 44.

There are many features in this data, but the most interesting one occurs at a running time of 20.0. Since the data begins on April 23 at 00:00 UT this feature corresponds to a time of 20:00 UT on April 23. This also corresponds to the peak of the geomagnetic storm shown in Figure 29 during which time Kp reached a value of 8. Because the Arduino magnetometer was in Kensington, Maryland, the closest magnetic observatory is FRD in Frederiksberg, VA. In Figure 30 we show both the FRD and Arduino data on the same plot for convenience.

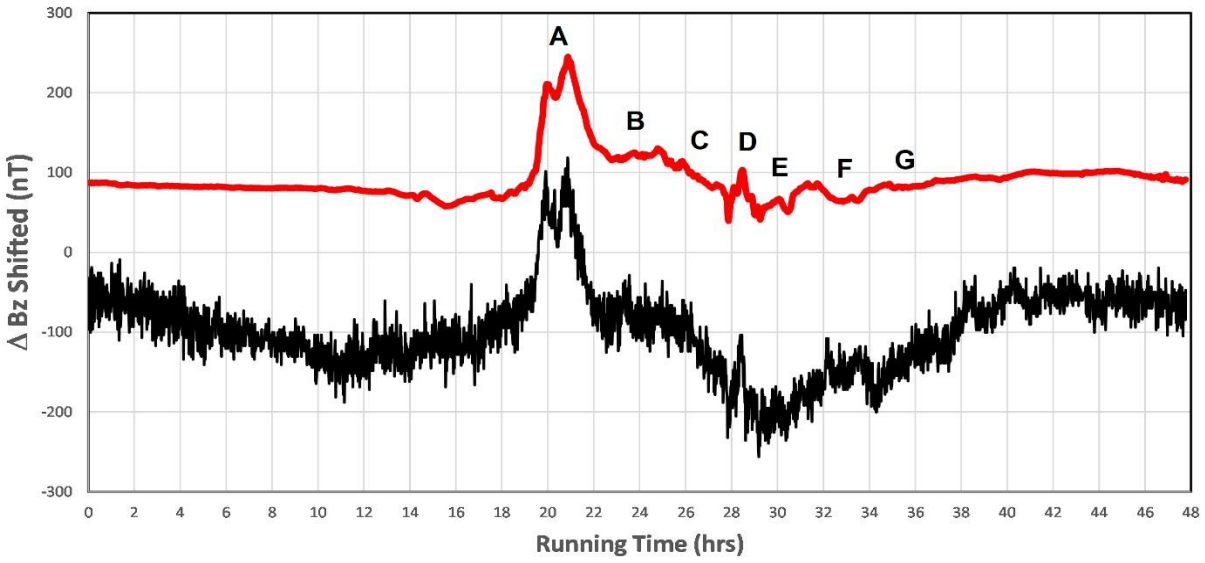


Figure 30. Comparison of FRD and Arduino data for the April 23 geomagnetic storm event.

The FRD data shows a series of features labeled A through F that generally match the corresponding features seen by the Arduino-RM3100 magnetometer. The double peaked maximum of the storm at 'A', is reproduced by the RM3100, as is the minor feature seen at 'D'. Based on the flat region in the data between 44.0 and 48.0, the noise (σ) in the FRD data is smaller than the width of the plotted red line with a value typically of about $\sigma = \pm 4.8$ nT. The corresponding noise in the one-minute cadence data from the RM3100 is about $\sigma = \pm 55.8$ nT. This is significantly higher than the nominal ±3 nT recorded under other conditions. The RM3100 data shows some significant systematic (oscillatory) variations with amplitudes of about 40 nT. Some of this may be due to the systematic variations seen at FRD that are contributing to its higher-than-expected noise near 5 nT.

Overall, the RM3100 magnetometer is fully able to detect major geomagnetic storms at mid-latitudes, and with an amplitude of about 200 nT for a Kp=8 event, weaker storms with Kp=5 should also be detectable at high S/N but with reduced amplitudes. At auroral latitudes of +55N and higher, this system should no doubt be capable of detecting many daily changes as ionospheric auroral currents wax and wane during these storm events.

Part VI. Detecting the diurnal Sq current

1. Background:

In the atmospheric region between about 85 and 200 km altitude on Earth, the ionospheric plasma is electrically conducting. Atmospheric tidal winds due to differential solar heating or due to the gravitational force of the Moon move the ionospheric plasma against the geomagnetic field lines, generating electric fields and currents just like a dynamo coil moving against magnetic field lines. That region is therefore called the ionospheric dynamo region.

The magnetic manifestation of these electric currents on the ground can be observed during magnetospheric quiet conditions. They are called Sq-variations (S=solar; q=quiet) of the geomagnetic field. More than 100 geomagnetic observatories around the world measure regularly the variations of the earth's magnetic field. The daily variations during selected days of quiet geomagnetic activity are used to determine a monthly mean.

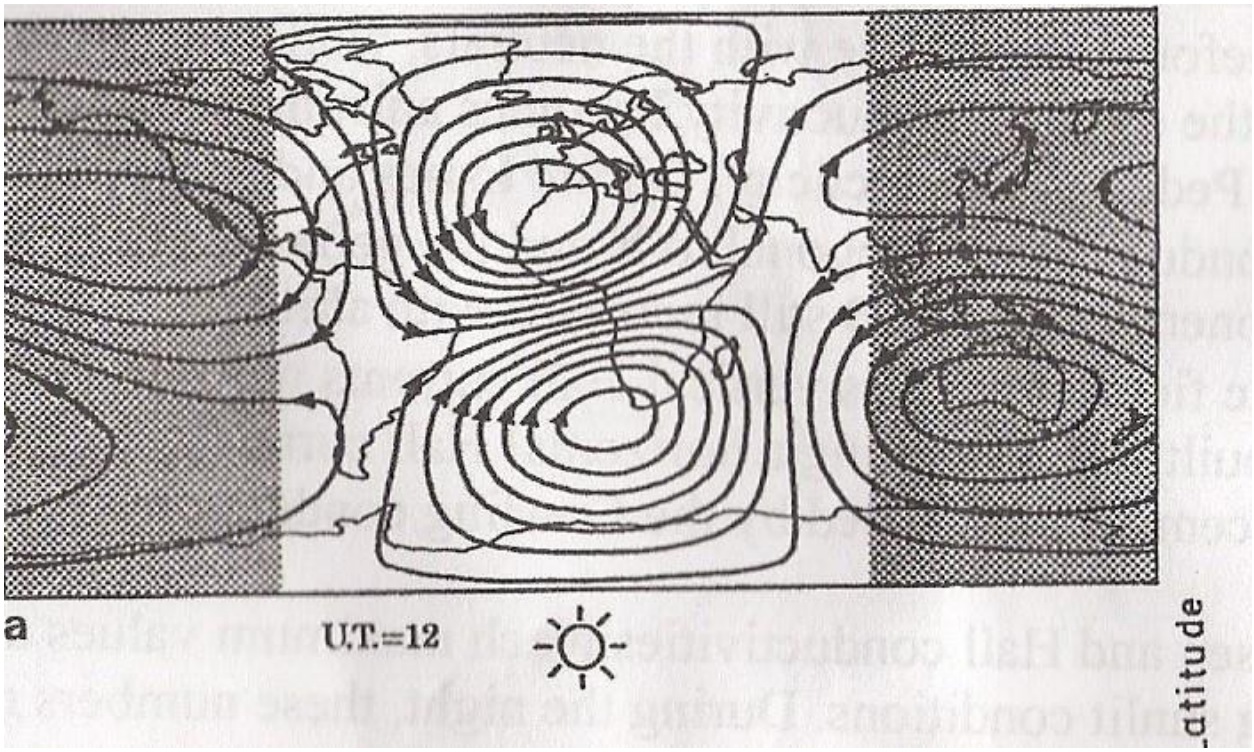


Figure 31. Diagram of Sq current in ionosphere (Credit: Wikipedia/Phil. Trans./Roy. Soc./S.R.C. Malin)

Figure 31 shows current streamlines of an equivalent Sq current as seen from the sun at noon. This current configuration is fixed to the sun, while the earth rotates beneath it. A total current of about 140,000 Amperes flows within one daytime vortex. In this example,

it is High Noon (UT 12:00) over the northwestern portion of Africa. If you were looking at the sun, there would be a circular ionospheric current flowing concentric with the solar disk and extending over a diameter of 10,000 kilometers east to west. Your local magnetometer would register a deviation in the local magnetic field of about 50 nT. As the sun rises and sets, this deviation would increase to a Noon maximum and then decrease to sunset, where your readings would return to the normal, undisturbed geomagnetic field value during the nighttime.

The Sq current depends on season. The summer vortex is intensified compared with the winter vortex and reaches into the winter hemisphere. During the 11-year sunspot cycle, the amplitude of Sq also increases by a factor of more than two from sunspot minimum to sunspot maximum.

Although strong geomagnetic storms occur on a weekly or monthly cadence, the diurnal Sq deviation can in principle be detected every day. It is a perfect 'test case' to determine just how sensitive the RM3100-based magnetometer is. To perform this detection will require patience and some data analysis, which we will now demonstrate.

2. Procedure

The approach we will be using is to set up the RM3100 system in a quiet location as far from magnetic disturbances (motors, computers, iron objects) as is possible. The location should be at room-temperature with a reasonable regulation to within ±1° F. The RM3100 is sensitive to changes in ambient temperature and these thermal changes can mask the very small effects we are looking for in the Sq deviation.

Figure 32. Raw data for Sq detection.

A continuous series of measurements were initiated on June 9, 2023, at 9:56pm EST (June 10 at 01:56 UT) and completed at 6:44am on June 13 (June 13 at 10:44 UT) providing a total of four days of measurements at a cadence of 1 minute per sample. The resulting raw data is shown in Figure 32 where the individual data segments were combined end-to-end to make a continuous time series. You will notice several jumps near 46.0 and 62.0 for example. Also, several glitches appear such as the one at 64.0. These are eliminated by masking out the glitches, which is done by substituting a local average value and removing jumps by simply adding a constant value to all subsequent data points until the jump is eliminated. The cleaned data is shown in Figure 33 with the jumps and other data artifacts removed.

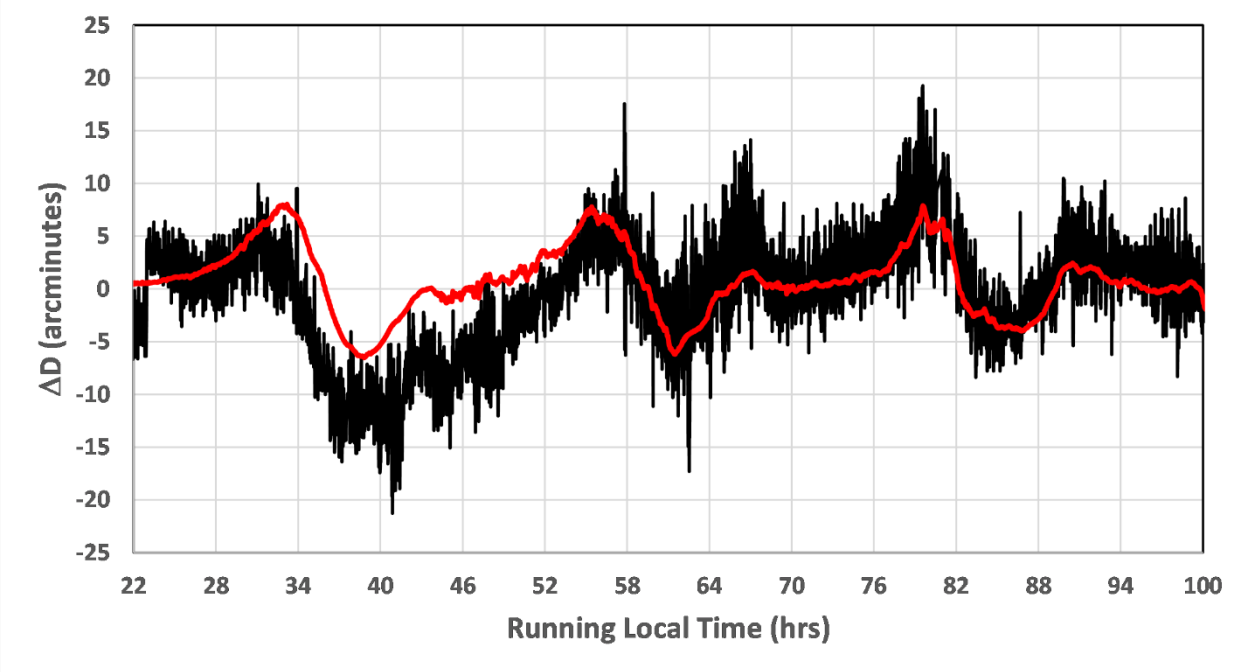


Figure 33. Cleaned data from the Arduino magnetometer. The FRD data is shown in red for comparison.

The RM3100 magnetometer data is rather noisy with a typical $\sigma = \pm 1.7$ arcmin, but there is a clear match between the FRD observatory data for the D-component and what the Arduino magnetometer records during the same period. Most of the differences between the Arduino and FRD data can be understood if there are additional small jumps in the Arduino data that have not been corrected in the cleaned data.

Part VII. Comparison of Smartphone and RM3100 data.

An iPhone 13 Pro smartphone and the RM3100 magnetometer were run simultaneously at a time when the geomagnetic conditions were at Kp=1 to compare their sensitivities and any internal artifacts in the data. A total of two hours of data were obtained with the smartphone *Physics Toolbox* app selected to run at its slowest cadence of 10 Hz, and the RM3100 code adjusted to generate a data point every 100 ms for a corresponding cadence of 10 Hz as well. The session generated two excel spreadsheets with 7200 x 10Hz = 72,000 rows of data.

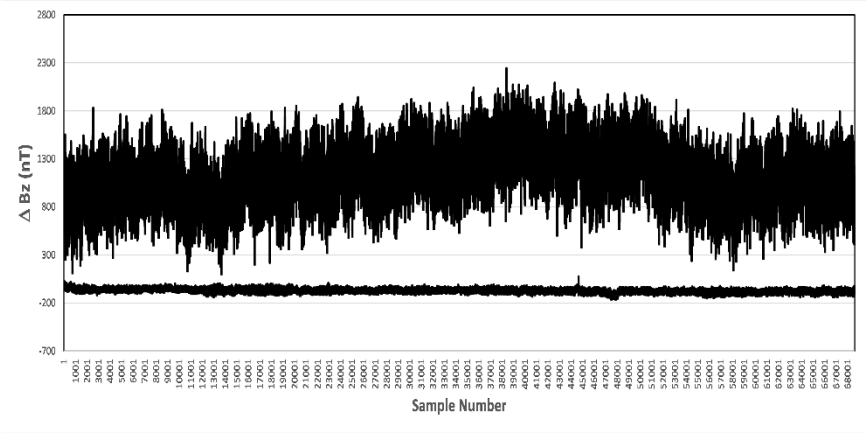


Figure 34. Combined Bz data for the Arduino (bottom) and iPhone 13 (top).

The Bz values were combined into one Excel worksheet with the 'Arduino' and 'iPhone' values in separate columns. These were then plotted on a single graph with the Arduino Bz data shifted uniformly by a constant additive factor so that the data appeared on the same plot. This is shown in Figure 34. A significant jump discontinuity occurred in the Arduino data near Sample 44653 (running time= 4465 seconds) so only the data prior to this event was used in the analysis and is replotted in Figure 35.

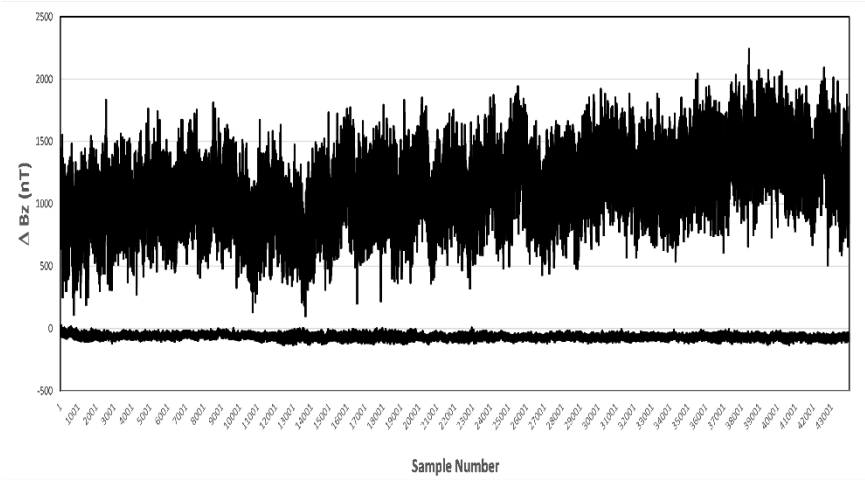


Figure 35. Data for the first 4465 seconds.

The first thing we notice is that the iPhone data has a persistent increase in baseline level over this period while the Arduino data is essentially flat. Baseline drifts with smartphones are not uncommon because the A/D converters being used to digitize the magnetometer data are usually temperature-sensitive, and are physically situated close to the battery, which heats up while under continuous use. You also will notice that the noise in the data is substantially higher for the smartphone than for the Arduino.

We can quantify the noise by computing the average and 'rms' or σ for the relatively constant-level data towards the end of the data series in Figure 36. The result of averaging Samples 41000 to 42000 is that for the Arduino we have σ = ±13.7 nT or equivalently in angular measure σ = ±1.2 arcminutes, while for the smartphone we have σ = ±209 nT. Another way to reveal any long-term trends is to average the 10 Hz data into 1-minute (60 sec x 10 Hz = 600) bins by using the Excel code =SUM(OFFSET(D2,(ROW()-1)*600,0,600,1))/600. The resulting plot is shown in Figure 36.

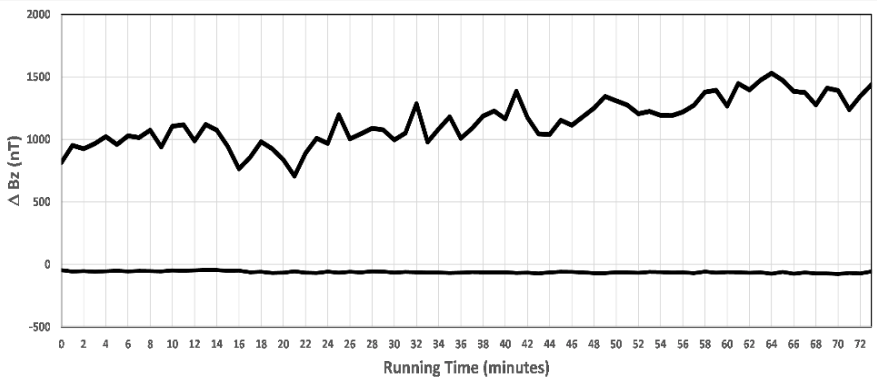


Figure 36. Bz data averaged into 1-minute samples for the smartphone (top) and the Arduino (bottom) magnetometers.

The progressing increase of the iPhone data is clear compared to the Arduino data. This comparison shows that the iPhone noise is ten-fold greater than the more-sensitive Arduino magnetometer, which makes the iPhone suitable for very strong geomagnetic events but not for the day-to-day or hourly changes of the geomagnetic field. In terms of the best places to site the Arduino magnetometer to further minimize external noise, the magnetometer was place in four different environments described in Table 6. The system took one measurement of Bz every minute for an hour. The indoor temperatures were well regulated to similar 'shirtsleeve' values however the outdoor temperature was significantly lower and apparently had a negative impact on the magnetometer noise.

Table 6. Data from different backgrounds.

Location	**Temperature**	**Average (μT)**	**Δ B (nT)**
Desk	68° F	39.2	23.4
Basement	66° F	35.8	21.1
Bedroom	68° F	38.7	23.2
Outdoors	50° F	41.1	42.8

National Aeronautics and Space Administration
Goddard Space Flight Center
9432 Greenbelt Road Greenbelt, MD 20771 https://nasa.gov/goddard/
www.nasa.gov

NP-2023-5-072-GSFC