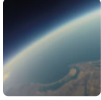




The Ultimate High Altitude Weather Balloon Data Logger



by AaronPrice

Record high altitude weather balloon data with the ultimate high altitude weather balloon data logger.

A high altitude weather balloon, also known as a high altitude balloon or HAB, is a huge balloon filled with helium. These balloons are a platform, allowing experiments, data collectors, or practically anything to go to near space. Balloons frequently reach heights of 80,000 feet with some going over 100,000 feet. A hab typically has a payload containing a parachute, radar reflector, and a package. The package usually

seminar about space. The seminar acted as inspiration. My peers decided they wanted to reach space. Touch the untouchable. They decided the way to reach space would be with weather balloons. Skip ahead four years later and we have launched 16 balloons. 15 have been recovered which is a very impressive track record for weather balloon retrievals. This year, I started high school and joined the weather balloon launching team. When I realized no data was being recorded, I set out to change that. My first data logger was [The Easiest Arduino High Altitude Balloon Data Logger](#). This new version captures

contains a camera and a GPS unit used to track and recover the balloon.

As the balloon gains altitude, the pressure drops. With less pressure outside the balloon, the balloon expands, eventually becoming so large, it pops! The parachute then returns the payload back to the ground, often many miles from where the balloon was launched.

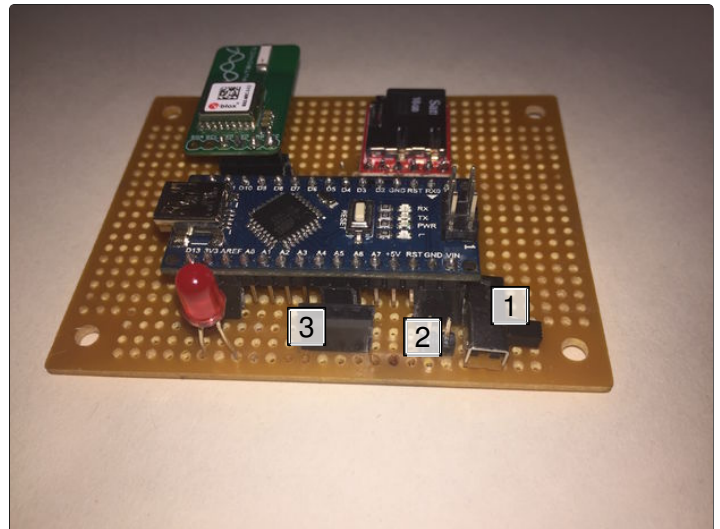
My school uses these balloons regularly to capture video of the curvature of the earth. With extreme temperature and pressure changes, large amounts of radiation, and wind speed, lots of interesting data can be captured from these flights.

This project began four years ago with a socratic



more data, earning it the title of ultimate. With this, altitude, temperature, wind speeds, ascent and descent rates, latitude, longitude, time, and date is captured and stored on a microSD card. This version also uses perf board to increase durability and lower risk. The design is made so that an Arduino Nano can be plugged in on top. The data collected from this data logger allows us, students to touch the edge of space. We can touch the untouchable!

This new data logger provides more data than most of the balloon loggers that can be purchased. It can also be built for less than \$80 while a store bought one will cost you more than \$200. Let's get started!



1. toggle on/off switch
2. for input power
3. Attaches to temp sensor +s-

<https://youtu.be/k-CAFmn5Vol>

Step 1: Parts, Programs, Tools, and Libraries

Parts

Arduino - A Nano is best as it can be snapped on top. I've also used Arduino Uno's with wires attaching it.

I would advise you to use a genuine Arduino because many of the clones may not work at the cold temperatures the data logger is exposed to. The coldest temp recorded on our flight was -58 fahrenheit. With proper weather protection and hand warmers, a clone can work.

\$5-\$22 (depending on quality)

<https://store.arduino.cc/usa/arduino-nano>

GPS Unit - This provides time, date, altitude, descent, ascent, and wind speed data!

I would highly recommend this unit. Most GPS units do not work above 60,000 feet. Since high altitude balloons go higher, those don't work. When in flight mode, this unit works to 160,000 feet.

https://store.uputronics.com/?route=product/product&product_id=72

\$30

MicroSD Data Logger - This holds a MicroSD card and allows us to store the data we collect.

There are many of these on the market and definitely

some for cheaper. I went with this one because it is lightweight, Sparkfun has great documentation, and it is very easy to use. When attached to pins 0 and 1, the Serial.print function writes to it. It's that easy!

<https://www.sparkfun.com/products/13712>

\$15

Temperature Sensor - I use one to provide outside temperature, but an additional can easily be added to provide temperature from inside the payload.

I used the tmp36 temperature sensor. This analog sensor runs without the delay command. The gps unit cannot work with delays therefore this sensor is ideal. Not to mention it is dirt cheap and only requires a single analog pin. Also, it works at 3.3 volts which is what the entire circuit runs on. This component is basically a perfect match!

https://www.sparkfun.com/products/10988?_ga=2.172610019.1551218892.1497109594-2078877195.1494480624

\$1.50

1k Resistors (2x) - These are used for the receiving lines of the GPS and MicroSD Data Logger.

The Arduino provides 5 volts to these pins. A 1k

resistor drops the voltage to a safe level for these units.

https://www.ebay.com/p/?iid=171673253642&lpid=82&&&ul_noapp=true&chn=ps

75¢

LED - This blinks every time data is collected (Optional).

The Arduino and MicroSd blink every time data is collected as well. This, however, makes it more obvious. The wires on this could be extended as well so that the led sticks out. This is used to ensure data logging is happening.

<http://www.ebay.com/itm/200-pcs-3mm-5mm-LED-Light-White-Yellow-Red-Green-Assortment-Kit-for-Arduino-/222107543639>

1¢

Perf Board - This allows a more permanent circuit and decreases risk as wires can't fall off. A breadboard or pcb could be used instead.

<https://www.amazon.com/dp/B01N3161JP?psc=1>

50¢

Battery connector - I use a 9v battery on my launches. This attaches the battery to the circuit. I solder the connection joint of jumper wires onto these to provide an easier connection.

<https://www.amazon.com/Battery-Connector-Plastic-A...>

70¢

Micro Toggle Switch - I use this to turn on the unit. This allows me to keep the battery plugged in while keeping the system turned off (Optional).

I salvaged mine from a moon lamp. Any micro switch will work.

[MicroSwitchLink](#)

20¢

Male and Female Headers - Use these to allow components such as the GPS and Arduino to detach from the circuit. (Recommended)

<http://www.ebay.com/itm/50x-40-Pin-Male-Header-0-1-2-54mm-Tin-Square-Breadboard-Headers-Strip-US-A-/150838019293?hash=item231ea584dd:m:mXokS4Rsf4dLAYh0G8C5RFw>

\$1

MicroSD Card - I would recommend a 4-16 gb card. The logs do not take up much space.

My data logger ran from 6:30 am to 1:30 pm and used only 88 kilobyte of space. That's less than 1/10 of a megabyte.

https://www.amazon.com/gp/product/B004ZIENBA/ref=oh_aui_detailpage_o09_s00?ie=UTF8&psc=1

\$7

Powersource - Space is cold so liquid batteries will freeze. This means no alkaline batteries. Lithium batteries work great! I used a 9v battery.

<https://www.amazon.com/Odec-9V-Rechargeable-Batter...>

\$1

Total Cost Comes In At \$79.66! Commercial loggers cost about \$250 so consider this a 68% discount. You also probably have many of these items such as the Arduino, Sd Card, etc which drive down the cost. Let's get to building!

Programs

The only program needed is the Arduino IDE. This is the native Arduino language and is used to upload the code, write code, and for testing. You can download the software for free here: <https://www.arduino.cc/en/Main/Software>

Libraries

We use two libraries in this sketch. The NeoGPS

library is used to interact with the GPS unit. The software serial library allows serial communication on additional pins. We connect to both the GPS and MicroSd data logger using serial communications.

NeoGPS

SoftwareSerial - Any software serial library can be used. I already had this one downloaded so I used it.

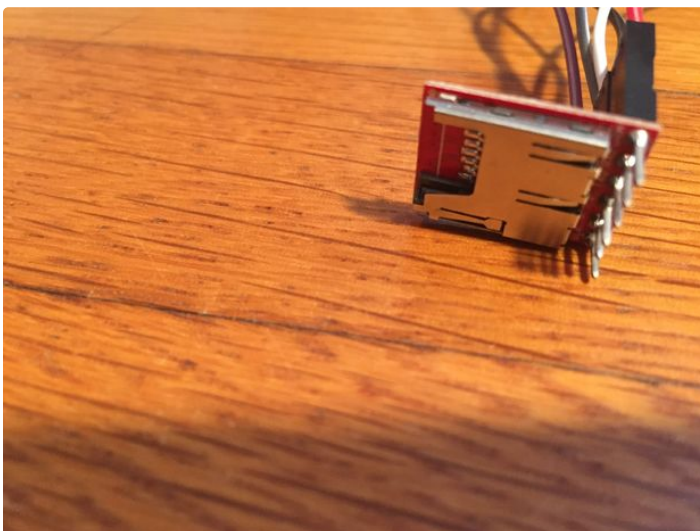
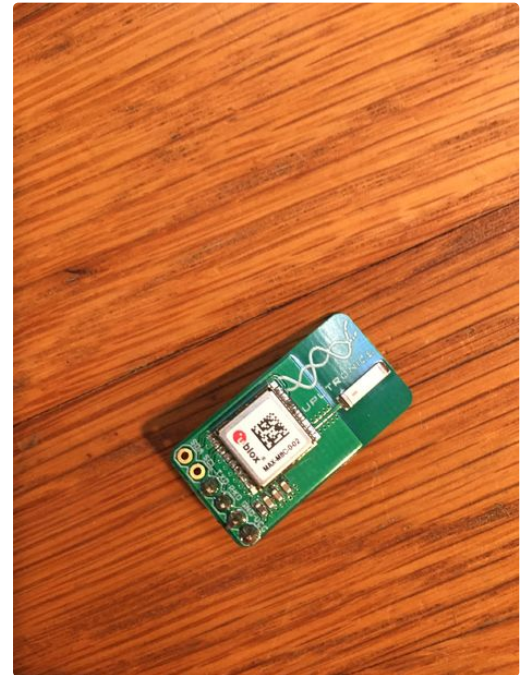
Need help installing a library? Read this:

<https://www.arduino.cc/en/guide/libraries>

Tools

Soldering Iron - Headers will need to be attached to multiple components and a soldering iron is used to attach components to perf board and make tracks.

Solder - Used in combination with soldering iron.





Step 2: Putting the Circuit Together

You will need to solder headers onto a few components. Learn how to do that here: <https://learn.sparkfun.com/tutorials/arduino-shield...>

Follow the breadboard or perf board schematic above. Do not attach ground of temp sensor to ground of GPS or microSD data logger as it will ruin your temperature data. If you are using a perf board, watch this tutorial on how to make tracks. This is one technique: <https://www.youtube.com/watch?v=3N3ApzmyjzE>

Be careful when attaching components. Make sure you have the correct polarity and pins. Check your connections twice!

3.3v ----- VCC

GND ---- GND

GND ---- BLK

Arduino -- Temp Sensor - Use the above photo to identify which leg is which

3.3v ----- VCC

GND ---- GND (This should either be on its own Arduino pin or attached to the power supply GND. If attached to GPS or logger it will skew temp data.)

Arduino - GPS

3.3v --- VCC

GND --- GND

D3 ----- 1k resistor----- RX

D4 ----- TX

Arduino -- OpenLog

Reset --- GRN

D0 ---- TX

D1 ---- 1k resistor----- RX

Signal --- A5

Arduino -- LED

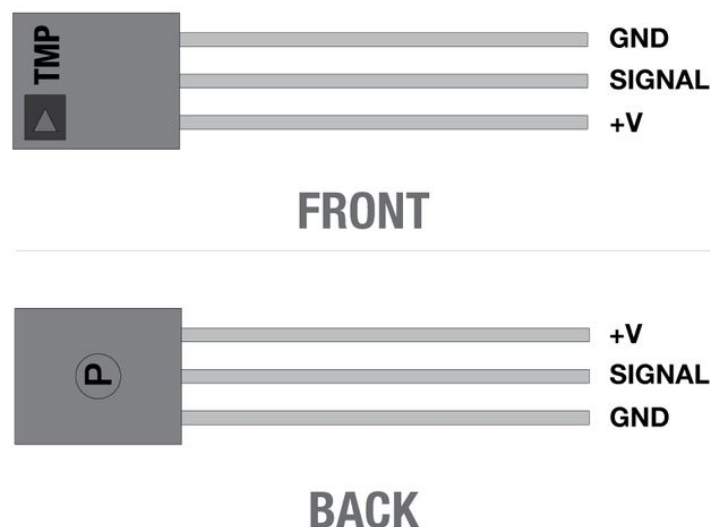
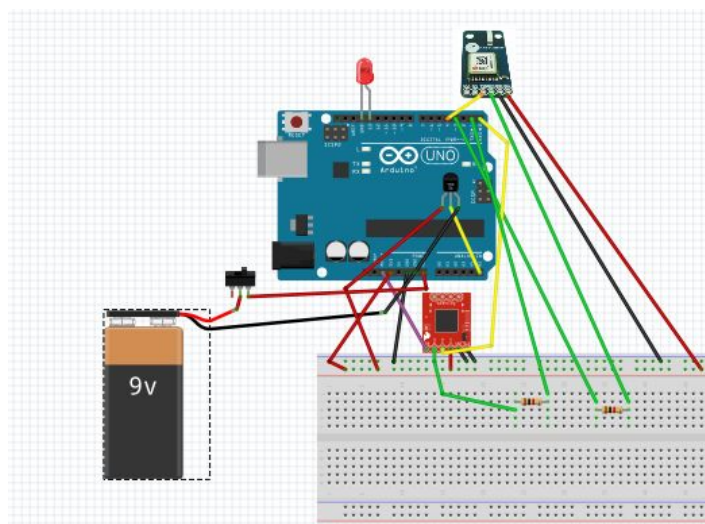
D13 ----- + (longer leg)

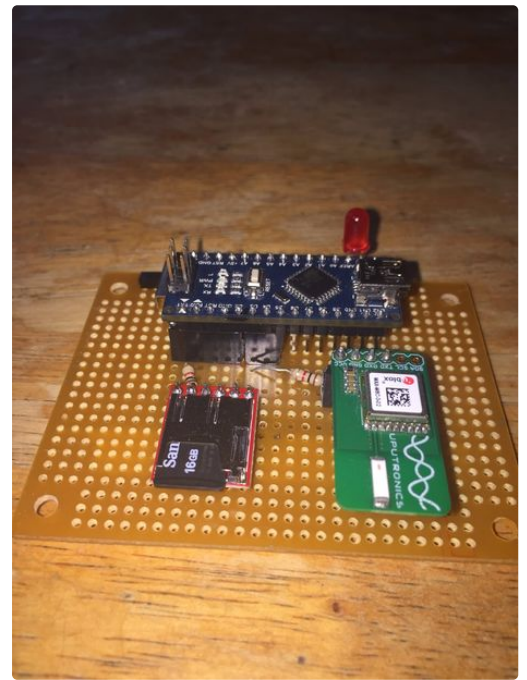
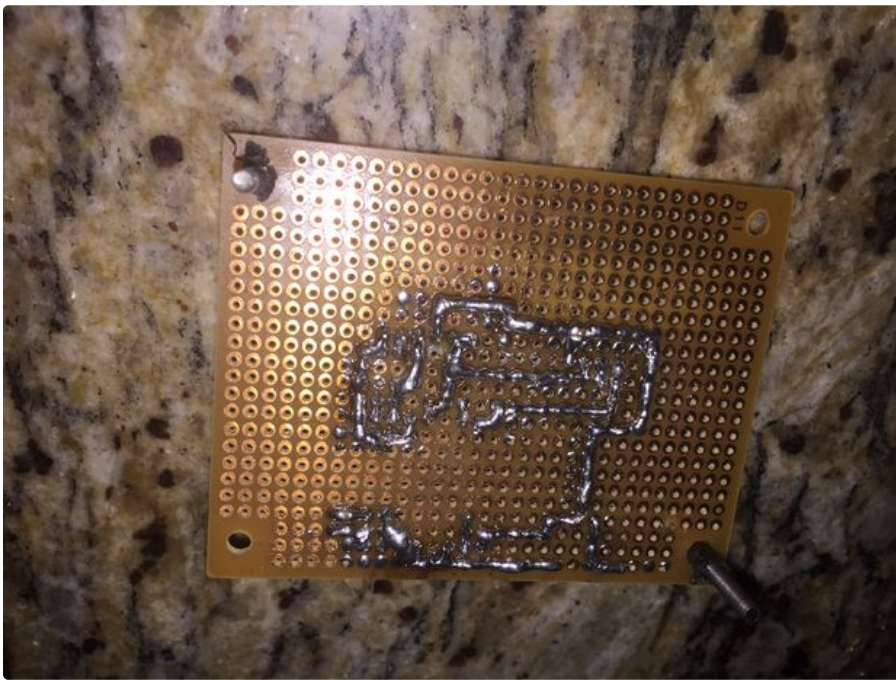
GND ----- - (shorter leg)

Arduino -- Battery Connector

Vin ---- Micro Toggle Switch ---- Positive(Red)

GND ----- Negative(Black)





Step 3: Programming

We use two libraries in this program, NeoGPS and SoftwareSerial. They can both be downloaded from the parts page of this Instructable. When interfacing GPS into an Arduino program, the TinyGPS library is typically used. However, I could not get it to work with the GPS we use.

The SoftwareSerial library allows us to connect two devices to the Arduino via software serial connection. Both the GPS and MicroSD data logger use this. Other libraries can do this as well and should work with the code. I already had this one on my computer and it works, so I used it.

The code is based off of my last data logger. The main change being the addition of the temperature sensor. GPS is based off of satellites. This means the GPS must first connect to satellites before it can display data. A lock consists of the GPS being connected to four satellites. A quick note being that the more satellites the GPS is connected to, the more accurate the data being provided is. The program prints the number of satellites connected on each line of data. It was connected to twelve satellites for most of my flight.

The program may need to be changed so that it works for you. While all of the code can be altered, I would recommend changing the timezone, time between readings, and unit of measurement for the temperature. A typical weather balloon is airborne for about two hours. The GPS receives data from the satellites every second. This means, if we store every piece of data sent, we will have 7,000 readings. Because I have no interest in graphing 7,000 data entries, I opt to log every 30th reading. This provides me with 240 data points. A little more reasonable of a number.

You may be wondering why we use a variable i and an if statement to save every 30th reading rather than just using a delay command and waiting 30 seconds. The answer is that GPS readings are very delicate. A 30 second delay means the GPS is not capturing every data set and causes our data to be messed up.

You'll need to change these values to your offset from Coordinated Universal Time (UTC).

If you don't know yours you can find it here <http://www.timeanddate.com/time/zone/offset.html>

```
static const int32_t
zone_hours = -8L; // PST

static const int32_t
zone_minutes = 0L; // usually zero
```

This line should be changed to how often you want a reading recorded. I set mine for a reading every 30 seconds.

```
if (i == 30) {
```

If you don't live in the U.S.A, you probably want temperature measurements in celsius. To do this, uncomment this line:

```
// Serial.print("Degrees C "); //uncomment if you want
celsius
```

```
// Serial.println(degreesC); //uncomment if you want
celsius
```

If you don't want fahrenheit readings, comment this:

```
Serial.print("Degrees F "); //comment if you do not
want fahrenheit
```

```
Serial.println(degreesF); //comment if you do not want
fahrenheit
```

Code Not Uploading?

The Arduino must be disconnected from the circuit while new code is being uploaded. The Arduino is sent the new code via Serial Communication on pins D0 and D1. These two pins are also the pins used for the MicroSd data logger. This means the MicroSD data logger must be disconnected for code to upload.

```

HighAltitudeBalloonV3.0
#include <Arduino.h>
#include <NMEAGPS.h>

//=====
// High Altitude Balloon Data Logger
// Written by Aaron Price
// Version 3.0
// Description: This program sets the GPS in flight mode then collects
// latitude, longitude, time, satellites locked, altitude, and wind speed.
// Update allows the collection of temperature data using TMP analog sensor on pin 0.
// It prints this data to the serial port or SD Card every 30 seconds.
// Either the onboard LED or the LED on D13 will blink every time data is recorded onto the SD card.
// Sketch uses NEOGPS library and is based off of the example sketch NMEAloc.
// Learn how to build the system here: https://www.instructables.com/id/The-Easiest-Arduino-High-Altitude-Balloon-Data-Log/
//=====

#if defined( UBRR1H ) || defined( ID_USART0 )
// Default is to use Serial1 when available. You could also
// use NeoHWSerial, especially if you want to handle GPS characters
// in an Interrupt Service Routine.
// #include <NeoHWSerial.h>
#else
// Only one serial port is available, uncomment one of the following:
// #include <NeoLSerial.h>
// #include <AltSoftSerial.h>
// #include <NeoSWSerial.h>
// #include <SoftwareSerial.h> // I used software serial
#endif
#include "GPSport.h"

```

```

HighAltitudeBalloonV3.0
#if !defined( GPS_FIX_SATELLITES )
#error You must uncomment GPS_FIX_SATELLITES in GPSfix_cfg.h!
#endif

#ifdef NMEAGPS_INTERRUPT_PROCESSING
#error You must *NOT* define NMEAGPS_INTERRUPT_PROCESSING in NMEAGPS_cfg.h!
#endif

#ifndef GPS_FIX_ALTITUDE
#error GPS_FIX_ALTITUDE must be defined in GPSfix_cfg.h!
#endif

//-----

static NMEAGPS gps; // This parses the GPS characters

const int temperaturePin = 5;

int i = 0;

byte gps_set_sucess = 0 ;

float voltage, degreesC, degreesF;

void sendUBX(uint8_t *MSG, uint8_t len) {
  for(int i=0; i<len; i++) {
    gps_port.write(MSG[i]);
    Serial.print(MSG[i], HEX);
  }
  gps_port.println();
}

```

```

HighAltitudeBalloonV3.0

#ifdef NeoHWSerial_h
#define Serial NeoSerial
#else
#define Serial Serial
#endif

//-----
// Check that the config files are set up properly

#if !defined( NMEAGPS_PARSE_RMC )
#error You must uncomment NMEAGPS_PARSE_RMC in NMEAGPS_cfg.h!
#endif

#if !defined( GPS_FIX_TIME )
#error You must uncomment GPS_FIX_TIME in GPSfix_cfg.h!
#endif

#if !defined( GPS_FIX_LOCATION )
#error You must uncomment GPS_FIX_LOCATION in GPSfix_cfg.h!
#endif

#if !defined( GPS_FIX_SPEED )
#error You must uncomment GPS_FIX_SPEED in GPSfix_cfg.h!
#endif

#if !defined( GPS_FIX_SATELLITES )
#error You must uncomment GPS_FIX_SATELLITES in GPSfix_cfg.h!
#endif

```

```

HighAltitudeBalloonV3.0
boolean getUBX_ACK(uint8_t *MSG) {
  uint8_t b;
  uint8_t ackByteID = 0;
  uint8_t ackPacket[10];
  unsigned long startTime = millis();
  Serial.print(" * Reading ACK response: ");

  // Construct the expected ACK packet
  ackPacket[0] = 0xB5; // header
  ackPacket[1] = 0x62; // header
  ackPacket[2] = 0x05; // class
  ackPacket[3] = 0x01; // id
  ackPacket[4] = 0x02; // length
  ackPacket[5] = 0x00;
  ackPacket[6] = MSG[2]; // ACK class
  ackPacket[7] = MSG[3]; // ACK id
  ackPacket[8] = 0; // CK_A
  ackPacket[9] = 0; // CK_B

  // Calculate the checksums
  for (uint8_t i=2; i<8; i++) {
    ackPacket[8] = ackPacket[8] + ackPacket[i];
    ackPacket[9] = ackPacket[9] + ackPacket[i];
  }

  while (1) {

    // Test for success
    if (ackByteID > 9) {
      // All packets in order!
      Serial.println(" (SUCCESS!)");
    }
  }
}

```

```

    return true;
}

// Timeout if no valid response in 3 seconds
if (millis() - startTime > 3000) {
    Serial.println(" (FAILED!)");
    return false;
}

// Make sure data is available to read
if (gps_port.available()) {
    b = gps_port.read();

    // Check that bytes arrive in sequence as per expected ACK packet
    if (b == ackPacket[ackByteID]) {
        ackByteID++;
        Serial.print(b, HEX);
    }
    else {
        ackByteID = 0; // Reset and look again, invalid order
    }
}
}
}

//-----
static void printL( Print & outs, int32_t degE7 );
static void printL( Print & outs, int32_t degE7 )
{
    // Extract and print negative sign

```

```

HighAltitudeBalloonV3.0
// Extract and print negative sign
if (degE7 < 0) {
    degE7 = -degE7;
    outs.print( '-' );
}

// Whole degrees
int32_t deg = degE7 / 1000000L;
outs.print( deg );
outs.print( '.' );

// Get fractional degrees
degE7 -= deg*1000000L;

// Print leading zeroes, if needed
int32_t factor = 1000000L;
while ((degE7 < factor) && (factor > 1L)){
    outs.print( '0' );
    factor /= 10L;
}

// Print fractional degrees
outs.print( degE7 );
}

static void doSomeWork();
static void doSomeWork( const gps_fix & fix )
{
    // This is the best place to do your time-consuming work, right after
    // the RMC sentence was received. If you do anything in "loop()",

```

```

HighAltitudeBalloonV3.0

Serial.print( ',' );
if (fix.valid.satellites)
    Serial.print( fix.satellites );

Serial.print( " , " );
Serial.print( fix.speed(), 6 );
Serial.print( " kn = " );
Serial.print( fix.speed_mph(), 6 );
Serial.print( " mph" );
Serial.print( " " );
Serial.print( gps_fix().altitude_cm());

} else {
    // No valid location data yet!
    Serial.print( '?' );
}

Serial.println();
digitalWrite(LED_BUILTIN, LOW);

i = 0;
}else{
    i = i + 1;
}

// doSomeWork

```

```

HighAltitudeBalloonV3.0

static const int32_t zone_hours = -7L; // PST
static const int32_t zone_minutes = 0L; // usually zero
static const NeoGPS::clock_t zone_offset =
    zone_hours * NeoGPS::SECONDS_PER_HOUR +
    zone_minutes * NeoGPS::SECONDS_PER_MINUTE;

NeoGPS::time_t localTime( fix.dateTime + zone_offset );

Serial << localTime;
}

if (fix.valid.location) {

    if ( fix.dateTime.seconds < 10 )
        Serial.print( '0' );
    Serial.print( fix.dateTime.seconds );
    Serial.print( ',' );

    // Serial.print( fix.latitude(), 6 ); // floating-point display
    // Serial.print( fix.latitudeL() ); // integer display
    printL( Serial, fix.latitudeL() ); // prints int like a float
    Serial.print( ',' );
    // Serial.print( fix.longitude(), 6 ); // floating-point display
    // Serial.print( fix.longitudeL() ); // integer display
    printL( Serial, fix.longitudeL() ); // prints int like a float

```

```

HighAltitudeBalloonV3.0

// the RMC sentence was received. If you do anything in "loop()",
// you could cause GPS characters to be lost, and you will not
// get a good lat/lon.
// For this example, we just print the lat/lon. If you print too much,
// this routine will not get back to "loop()" in time to process
//the next set of GPS data.

// test vs other option
//float voltage, degreesC, degreesF;
//voltage = getVoltage(temperaturePin);
//degreesC = (voltage - 0.5) * 100.0;
//degreesF = degreesC * (9.0/5.0) + 32.0;
//Serial.println(degreesF);

if (i == 30) { // keeps a reading every 30 seconds. Change this to the time wanted between readings

    //concern tha this takes time therefore could throw off repetition
    // consistency
    // float voltage, degreesC, degreesF;
    // voltage = getVoltage(temperaturePin);
    // degreesC = (voltage - 0.5) * 100.0;
    // degreesF = degreesC * (9.0/5.0) + 32.0;
    // Serial.print("Degrees C "); //uncomment if you want celsius
    // Serial.println(degreesC); //uncomment if you want celsius
    Serial.print("Degrees F "); //comment if you do not want fahrenheit
    Serial.println(degreesF); //comment if you do not want fahrenheit

    digitalWrite(LED_BUILTIN, HIGH);
    if (fix.valid.time) {
        // Set these values to the offset of your timezone from GMT
        static const int32_t zone_hours = -7L; // PST
    }
}

```

```

HighAltitudeBalloonV3.0

}

float getVoltage(int pin)
{
    return (analogRead(pin) * 0.004882814);
}

//-----

static void GPSloop();
static void GPSloop()
{
    while (gps.available( gps_port ))
        doSomeWork( gps.read() );
} // GPSloop

//-----

void setup()
{
    Serial.begin(9600);

    pinMode(LED_BUILTIN, OUTPUT);

    // Start the UART for the GPS device
    gps_port.begin(9600);

    Serial.println("Setting uBlox flight mode: ");
    uint8_t setNav[] = {
        0x85, 0x62, 0x06, 0x24, 0x24, 0x00, 0xFF, 0xFF, 0x06, 0x03, 0x00, 0x00, 0x00, 0x00, 0x10, 0x27, 0x00, 0x00,
        0x05, 0x00, 0xFA, 0x00, 0xFA, 0x00, 0x64, 0x00, 0x2C, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    };

```

HighAltitudeBalloonV3.0

```
0x05, 0x06, 0x06, 0x24, 0x00, 0xFF, 0xFF, 0x06, 0x03, 0x00, 0x00, 0x00, 0x10, 0x27, 0x00, 0x00,
0x05, 0x00, 0xFA, 0x00, 0xFA, 0x00, 0x64, 0x00, 0x2C, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x16, 0xDC };
while(!gps_set_success)
{
    sendUBX(setNav, sizeof(setNav)/sizeof(uint8_t));
    gps_set_success=getUBX_ACK(setNav);
}
gps_set_success=0;

while (!Serial);

Serial.print( F("IMEIloc IN0: started\n") );
Serial.print( F("fix object size = ") );
Serial.print( sizeof(gps.fixC) );
Serial.print( F("IMEAGPS object size = ") );
Serial.print( sizeof(gps) );
Serial.print( F("Looking for GPS device on " USING_GPS_PORT) );
Serial.print( F(" by Aaron Price");
Serial.println();
Serial.print( " Time Latitude Longitude SAT Wind Speed Wind Speed Altitude");
Serial.print( " (deg) (deg) knotts mph cm");
Serial.println("-----");

#ifdef IMEAGPS_NO_MERGING
    Serial.println( F("Only displaying data from xxRMC sentences.\n Other sentences may be parsed, but their data will not be displayed.") );
#endif
Serial.flush();
```

HighAltitudeBalloonV3.0


```
Serial.println( sizeof(gps) );
Serial.println( F("Looking for GPS device on " USING_GPS_PORT) );
Serial.print( "High Altitude Weather Balloon Data Logger");
Serial.println( F(" by Aaron Price");
Serial.println();
Serial.print( " Time Latitude Longitude SAT Wind Speed Wind Speed Altitude");
Serial.print( " (deg) (deg) knotts mph cm");
Serial.println("-----");

#ifdef IMEAGPS_NO_MERGING
    Serial.println( F("Only displaying data from xxRMC sentences.\n Other sentences may be parsed, but their data will not be displayed.") );
#endif
Serial.flush();

}


//-----
void loop()
{
    GPSloop();

    // If the GPS has been sending data, then the "fix" structure may have
    // valid data. Remember, you must check the valid flags before you
    // use any of the data inside "fix". See "dataElements" for an example
    // of checking whether any lat/lon data has been received yet.
}
```



<https://www.instructabl...>

Download



<https://www.instructabl...>

Download

Step 4: Testing

Once all connections are made and the code is uploaded, it is time to test our data logger. To do this, plug the Arduino into the computer the same way you would to upload code. Ensure the serial port is correct and then open the Serial Monitor. If all connections are made correctly, this will be displayed:

NMEALoc.INO: started
fix object size = 31 NMEAGPS object size = 84
Looking for GPS device on SoftwareSerial(RX pin 4, TX pin 3) High Altitude Weather Balloon Data Logger
by Aaron Price

Time Latitude Longitude SAT Wind Speed Wind
Speed Altitude (deg) (deg) knots mph cm -----

If the GPS is plugged in incorrectly, this will be displayed:

Setting uBlox flight mode:
B562624240FFFF63000010270050FA0FA06402C10
0000000000000016DC * Reading ACK response:
(FAILED!)

Ensure the led blinks every time a new piece of data enters the Serial Monitor. The MicroSd data logger will also blink every time data is recorded.

You will notice that the GPS is sending you a single question mark. This is because GPS units take time to start up and connect to satellites. This unit typically takes about eight minutes to start sending me the full string of data. Within about five it will begin sending you date and time data followed by a question mark. The first few points will probably be incorrect but then it will display the correct date and time. If you are not receiving your date and time, refer to the code to ensure the right timezone is corrected. Read the programming section of this Instructable to learn how to do that.

Eventually the Serial Monitor will display all of the data. Copy and paste the latitude and longitude and prepare to be shocked by the results. The accuracy is

remarkable!

Check the temperature data to make sure it is correct. If the temperature is being read as a grossly unrealistic number (160+) the temperature sensor is either not plugged in or plugged in incorrectly. Refer to the schematic. If the temperature reading is volatile or higher than it should be (i.e. the temperature is 65 fahrenheit and the sensor reports it as 85) then the sensor is likely sharing a ground pin with the GPS, microSD data logger, or both. The temp sensor should either have its own ground pin or share a ground pin with only the input ground.

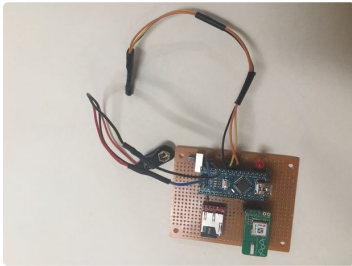
You now need to format and clear your microSD card. We need a fat16 or fat32 file type. I followed this tutorial by GoPro: https://gopro.com/help/articles/Solutions_Troubleshooting/SD-Card-Reformat-on-a-Windows-Computer

Next, test the circuit without the computer attached. Plug a microSD card into the data logger and use a power source to give the Arduino power. Let it run for twenty minutes then disconnect power. Unplug the microSD card and plug it into your computer. You should see a config file was created (this only happens when a prior config file is not made). Each time the Arduino is reset or plugged in, it creates a new file.

New libraries and versions of the Arduino IDE have been released since the conception of this project. Because of this, multiple users were getting nasty error messages. User RahilV2 was having this issue and found a solution!

"I have fixed the initial error and it was because the .INO uses the old gps port name which is 'gpsPort' instead of 'gps_port'. The preprocessor symbol has also changed. All the example programs now use 'GPS_PORT_NAME' instead of 'USING_GPS_PORT'."

Thanks RahilV2!



```
Setting uBlox flight mode:
B562624240FFFF63000010270050FA0FA06402C10000000000000016DC * Reading ACK response: (FAILED!)
B562624240FFFF63000010270050FA0FA06402C10000000000000016DC * Reading ACK response: (FAILED!)
B562624240FFFF63000010270050FA0FA06402C10000000000000016DC * Reading ACK response: (FAILED!)
B562624240FFFF63000010270050FA0FA06402C10000000000000016DC * Reading ACK response: (FAILED!)
B562624240FFFF63000010270050FA0FA06402C10000000000000016DC * Reading ACK response:
```

1. if this appears, check GPS connections

```
Setting uBlox flight mode:
B562624240FFFF63000010270050FA0FA06402C10000000000000016DC * Reading ACK response: B5625120624325B (SUCCESS!)
NMEAloc, IND: started
fix object size = 31
NMEA GPS object size = 84
Looking for GPS device on SoftwareSerial( RX pin 4, TX pin 3 )
High Altitude Weather Balloon Data Logger by Aaron Price
```

Time	Latitude (deg)	Longitude (deg)	SAT	Wind Speed knotts	Wind Speed mph	Altitude cm
------	-------------------	--------------------	-----	----------------------	-------------------	----------------

1. This means the GPS is hooked up correctly, prepare to get readings!

```
Setting uBlox flight mode:
B562624240FFFF63000010270050FA0FA06402C10000000000000016DC * Reading ACK response: B5625120624325B (SUCCESS!)
NMEAloc, IND: started
fix object size = 31
NMEA GPS object size = 84
Looking for GPS device on SoftwareSerial( RX pin 4, TX pin 3 )
High Altitude Weather Balloon Data Logger by Aaron Price
```

Time	Latitude (deg)	Longitude (deg)	SAT	Wind Speed knotts	Wind Speed mph	Altitude cm
------	-------------------	--------------------	-----	----------------------	-------------------	----------------

```
Degrees F 70.32
?
Degrees F 70.32
?
```

1. The temp sensor works right away. The GPS requires time before it can provide data.

```
Setting uBlox flight mode:
B562624240FFFF63000010270050FA0FA06402C10000000000000016DC * Reading ACK response: B5625120624325B (SUCCESS!)
NMEAloc, IND: started
fix object size = 31
NMEA GPS object size = 84
Looking for GPS device on SoftwareSerial( RX pin 4, TX pin 3 )
High Altitude Weather Balloon Data Logger by Aaron Price
```

Time	Latitude (deg)	Longitude (deg)	SAT	Wind Speed knotts	Wind Speed mph	Altitude cm
?						
Degrees F 70.32						
?						
Degrees F 70.32						
2017-06-13 20:34:18?						
Degrees F 70.32						
2017-06-13 20:34:49?						
Degrees F 70.32						
2017-06-13 20:35:20?						
Degrees F 70.32						
2017-06-13 20:35:51?						
Degrees F 70.32						

1. The GPS will then begin providing time and date data.

```
Degrees F 70.32
2017-06-09 09:05:4646,32.9752030,-116.8112045,11 , 0.086000 kn = 0.098967 mph
62620
Degrees F 70.32
2017-06-09 09:06:1717,32.9752030,-116.8112083,12 , 0.182000 kn = 0.209442 mph
62810
Degrees F 70.32
2017-06-09 09:06:4848,32.9752367,-116.811257,-12 , 0.054000 kn = 0.062142 mph
3900
```

1. This is what a full string of our data looks like.
2. This is the number of satellites in contact. The more, the more accurate the reading
3. This is the altitude in centimeters. When the altitude is a large number, it is displayed here.

Step 5: Protecting the Electronics

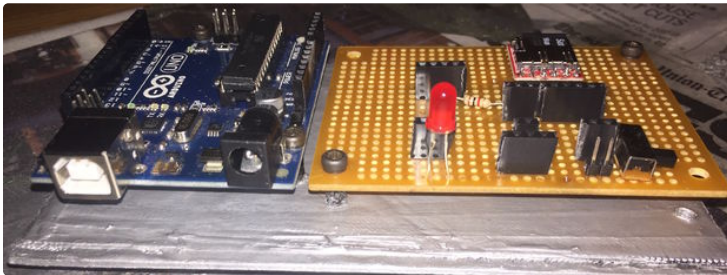
A note to people using perf board, placing the circuit on a metal surface will short the circuit. I used a plastic pipe around some bolts to hang my perf board above a plastic sheet. You could hot glue the bottom, attach it to some cardboard or foam, or use a package that does not conduct electricity. You can 3d print these plastic pipes to slip over your bolts from here: <https://www.thingiverse.com/thing:2382639>

I attached female headers to the perf board where the GPS sits to allow the GPS to be easily snapped off the circuit. The GPS unit is fragile. The chip antennae can break and the unit is sensitive to static electricity. I haven't had any of these units break. I store the GPS in the static shielded bag it comes in to keep the GPS protected.

Whether you are using a breadboard or just jumper wires for the battery connector, I recommend using

hot glue to ensure the jumper wires stick in their sockets. It would be a bummer for you to recover your balloon to find it didn't log because a jumper wire detached.

Hand warmers are advised as they will keep everything warm and functioning. I typically extend the length of my battery connectors allowing me to store the battery in a separate compartment from the electronics. I put hand warmers directly on the battery. While the electronics should be able to function without hand warmers, I would recommend using them. Put a hand warmer or two near the electronics, securing the hand warmer so that it will not touch the electronics. The radiant heat from the hand warmers is sufficient in keeping the electronics in good condition.



Step 6: Launch

I typically plug the data logger into my computer about twenty minutes before we plan to let the balloon go. Plugging the logger into the computer is not necessary. I do this to ensure the GPS is running and that I have a satellite lock. Once the logger is displaying all of the data, I flip the toggle switch and disconnect the computer. Because the circuit always has a source of power, the GPS stays hot and continues logging with a satellite lock. This will create a new file on the microSD card.

We launched the balloon at 6:58 am. We planned to launch earlier but our first balloon developed a rip. We had forgotten our tubing to attach the balloon to the helium tank. So, we attached the balloon directly to the nozzle of the helium tank. The vibrations on the

nozzle put a rip in the balloon. Luckily, we brought a spare balloon. We used a cut garden hose as our improvised tubing and it worked!

The package consisted of an insulated lunchbox. The data logger sat inside with the hand warmers. A hole cut in the lunchbox provided a way for the camera to be inside the lunchbox while maintaining an unobstructed view. We used a GoPro Session for this launch. It took photos of the journey! Attached to the side and top of the lunch box were two SPOT GPS units. We used these to track our package. A small slit was made in the side of the lunchbox to allow the temperature sensor to stick out, exposing it to the outside air.



1. filling up the balloon. We use a 600 gram balloon
2. filling up the balloon. We use a 600 gram balloon



1. This photo was taken from 8,000 feet.
2. The famous Crystal Pier
3. Crystal Pier opened in 1927

<https://youtu.be/TZW0-AfJt7w>

Step 7: Recovery

I used a Duracell 9v battery on my last launch. I measured the voltage of the battery as 9.56 volts before plugging it into the data logger. I plugged the battery in at around 6:30 am. After the balloon landed, was recovered, driven back to school, and the package opened, it was 1:30 pm. I opened up the payload to find the data logger was still logging! I then measured the voltage of the 9v battery. As a battery is used, the voltage lowers. The battery was now at 7.5 volts. After seven hours of data logging, the battery was still in decent shape.

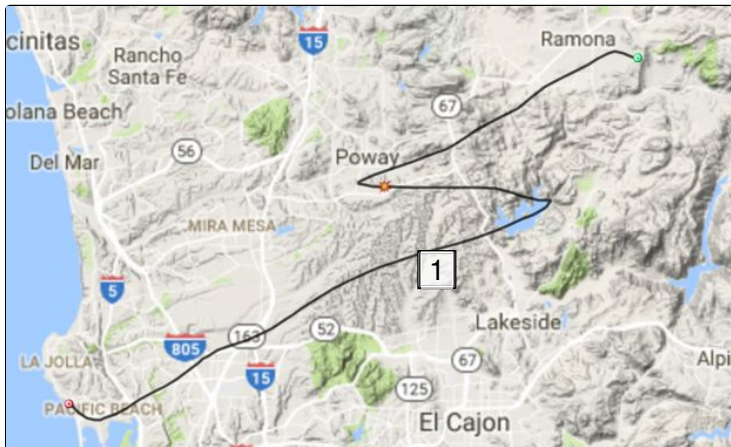
The balloon and package landed south of Ramona in a small canyon. The recovery team drove about an hour and then hiked the rest of the way. Poison ivy and hot temperatures were an obstacle, yet they persevered and were able to recover the balloon. They returned to the school and handed me the

package. I was surprised the data logger was still running. This made me optimistic. I unplugged the battery and carefully took out the microSD card. I then ran to my computer. This is the most nerve racking and exciting part of the journey for me. Did the data logger work? I rummaged through my backpack to find the SD card adapter. The last two flights the logger had stopped working at 40,000 feet because I had incorrectly put the GPS in flight mode. Being that the only way I can reach heights over 40,000 feet is with weather balloons, I had no idea if my new code would work.

I plugged the microSD card into my computer, opened the file, and saw a log full of data. I began scrolling through the data... SUCCESS!! The log continued through the entire flight.



1. This was the actual path of travel. We use SPOT GPS to track our package.



1. We use predict.habhub to give us a prediction of where the balloon will travel. We use this to ensure the balloon will be retrievable.

Step 8: Analysis and Science

The expression "third times the charm" rings true. We logged data for the entire flight! The balloon reached a max altitude of 91,087 feet and the coldest temperature was -58 degrees fahrenheit.

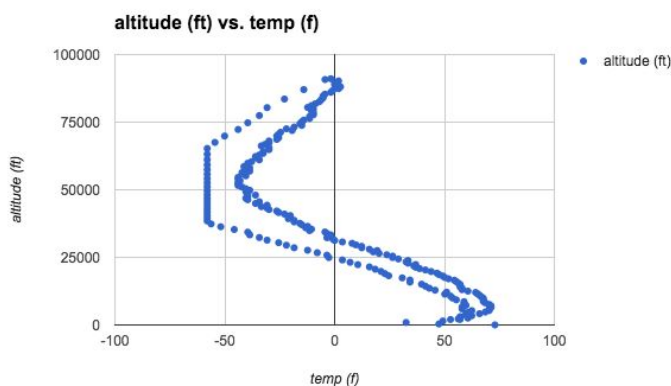
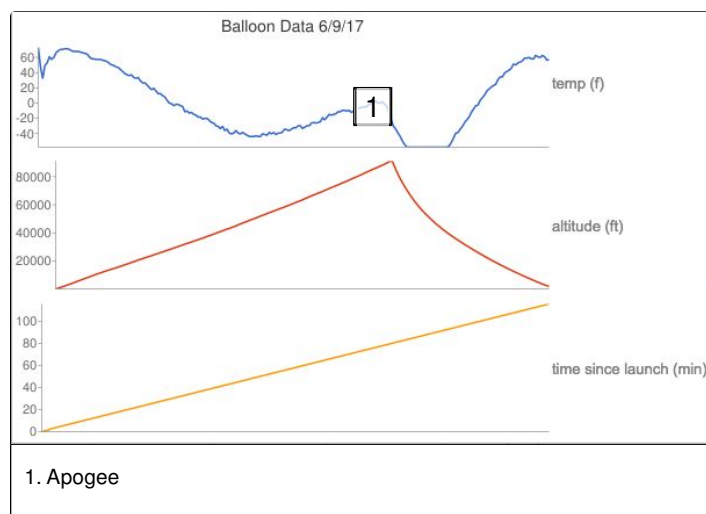
Our data confirms and aligns with much of known science. For example, the bottom of the stratosphere was -40 to -58 degrees fahrenheit while at apogee of the flight, the temperature was -1.75 degrees fahrenheit. Humans live in the lowest layer of the Earth's atmosphere, the troposphere. In the troposphere, the temperature lowers as one gains in altitude. The opposite is true in the stratosphere. In fact, the top of the stratosphere can be five degrees above zero.

I was surprised that the balloon ascended in such a linear fashion. I would think as the atmosphere thinned the balloons ascent speed would change. I was not surprised, however, by the curve in the descent speed of the balloon. My hypothesis as to why the balloon falls quickly then gradually slows down has to do with the parachute. At apogee, there is so little air that I think the parachute was not as effective. Parachutes use air resistance and friction to slowly fall to the ground so if there is little air, the

parachute is not as effective. As the package lowers, the air resistance increases because there is more air pressure and more air. This results in the parachute being more effective and the package descending slower.

Due to temperature and wind speeds, I declare the worst altitude to live in to be 45,551 feet. At this altitude, the package experienced a chilly -58 degrees fahrenheit. If this was not enough, winds were blowing 45 miles per hour. While I had trouble finding data for the effect of wind on windchill at this temperature, I found that -25 degree fahrenheit weather with a 45 mile per hour wind results in a windchill of -95 degrees. I also discovered that windchill temperatures of -60 degrees freeze exposed flesh in 30 seconds. Nonetheless, this is probably not an ideal vacation spot. As seen in the photo above, there is a great view from this altitude! Learn more about windchill here: <http://www.math.wichita.edu/~ric hardson/windchill.....>

I could not have displayed and studied this data without help from my sister who did data entry of all 240 lines of data. Perks of having younger siblings :)





1. With -58 degree fahrenheit temps and 45 mile per hour winds, 45,551 feet isn't the best beach weather. There is a really nice view however.



<https://www.instructabl...>

Download

Step 9: Conclusion

This is a definite success. We logged altitude, temperature, wind speed, ascent rate, descent rate, time, date, latitude, and longitude data on the entire flight. This is a must have for experienced high altitude ballooners and first time launchers!

After four years of balloon launching, we finally data logged an entire flight. We finally found out how high our balloons fly. We got a little closer to experiencing space. We got a little closer to touching the untouchable!

Another cool aspect of the data logger is that all the data is time stamped. This means you can line up the data with photos taken on the journey which allows you to know the altitude and exact location in which each photo was taken!

This project is easy to replicate and modify for your own purposes. Easily add additional temperature sensors, pressure and humidity sensors, geiger counters, the opportunities are endless. As long as the sensor can be used without a delay, it should work!

Thank you for taking time to read this Instructable. I enjoy answering questions, replying to comments, and helpful tips and ideas, so shoot away in the comments section below.

This Instructable is also in some contests, please vote if you enjoyed or learned something new! Winning prizes allows me to earn new tools to make better and more advanced projects.



<https://youtu.be/k-CAFmn5Vol>



Hi, great project, and really well documented, great job.

I'm hoping to piggy back on your project as I want to run a couple of experiments (self funded) at and around 100,000ft. I need to get a number of high accuracy temperature measurements, so I bought 5 DS18B20 digital temperature sensors,
<https://www.amazon.co.uk/gp/product/B075FYLLV/ref...>

Unfortunately these have a delay which, like you say, gets in the way of the GPS monitoring, do you know of a way to get around this or is the only solution to have a second datalogger (Arduino, SD card, etc.)?

Thanks very much

Rob



Hi Aaron, great job. I have some questions to ask you.

I bought the gps you recommended. Can I use the TinyGPS ++ library to make it work over 120000 feet? Or does it crash at 60000 feet? I downloaded your code, but I would only need the part to detect altitude and coordinates. I do not need time and temperature.



I'm having trouble with the program itself. I've been troubleshooting for awhile now, and I keep receiving an error in the arduino IDE telling me that gps_port is not declared. Is there a library that I could be missing where it is declared? Do you have any ideas on a solution? Any help would be most appreciated :)



Awesome to see you are working on the project. I have been receiving quite a few emails lately regarding this exact issue. RahilV2 built the data logger and was able to solve the problem. Here is what he did:

I have fixed the initial error and it was because the .INO uses the old gps port name which is 'gpsPort' instead of 'gps_port'. The preprocessor symbol has also changed. All the example programs now use 'GPS_PORT_NAME' instead of 'USING_GPS_PORT'.

It seems all you will need to do is change the names of a few values as Rahil explains above. Hope that helps!



Ensure you have the softwareSerial library and NeoGPS library downloaded.



This is a great project but is there any security issues with flying object? The result is great.



If the flight is cleared with the FAA and a radar reflector is attached to the payload, you are all good!



Great job on this instructable! Strong with the force you are young Jedi!



Thank you!



Awesome project dude! I'm launching a balloon in a few weeks by myself and this is perfect! In my iteration, I also plan to add a Geiger counter to measure the change in radiation. I feel that data would be pretty cool.

I do have two questions.

1. Do you know why that particular GPS works above 60,000ft and why others do not?
2. How did you track the damn thing during recovery? I've been trying to find an economical solution for a week now, I haven't been able to find much that is not a HAM radio that costs 300\$.

The best I've got is a GPRS tracker but that depends heavily on cell reception, and that isn't reliable around where I live.

Again, really cool project!



Most GPS units do not work above 60,000 feet to stop people from launching ICBMs (intercontinental ballistic missiles). Cool right?! GPS units must shut down if the unit is moving 1200 mph or is above 60,000 feet. A lot of GPS manufacturers have the GPS shut down if either the GPS is moving at 1200 mph or the unit is above 60,000 feet. The unit I use will only shut down if the unit is moving at 1200 mph and is above 60,000 feet. Since we never exceed 1200 mph, the unit continues to work.

We use the SPOT GPS system to track our balloons. <https://www.findmespot.com/en/> We have had very good results with this system. We attach two units to every payload in case one stops working. These units send location data to us in realtime via text message and an online map. I think the SPOT GPS system works without the need for cell reception.

Hope this helps!



ah that is pretty cool indeed



Outstanding! It is, in a way, unfortunate that weather balloons typically burst at the altitude you mention. If they were tougher, you could try for the Karman line (100 km, or about 330,000 ft) which is defined as the lower edge of outer space.



Of course the toughness of the balloon has some effect. I think the easiest way to achieve higher altitudes would be to use a larger balloon, maybe 1200 gram or larger and underfill it. With less helium inside, it will take a more substantial drop in pressure for the balloon to expand to a point in which it pops. The Karman line would definitely be an exciting milestone. I'm wondering though if we would have a better shot with amateur rocketry.



exceeding the Karman line is extremely hard, even with amateur rocketry.

I believe only 2 amateur groups, (I say groups but they are more like companies) have crossed the Karman line.



Very curious! I like it.



Thanks!



I only just started on the article and I noticed that you linked to Amazon alkaline batteries when you stated that space is too cold for them. Was this a simple mistake after you promoted Lithium as ideal? I guess alkaline will work from inside a well-insulated case (and they're a lot cheaper than even the non-rechargeable "9V" Lithiums).



Good catch. That was a mistake, fixed now. With a well insulated package, any battery type will work. If exposed to the open however, alkaline batteries should not be used. I actually used an alkaline battery on my last launch. They are cheap and accessible. I had a very well insulated package with hand warmers as well.



I expected that insulation was an "inexpensive" option. So, did you do a log of the internal temperature on any of those launches that used insulation and then compared it to the external temperature to see how effective the insulation was?



Unfortunately, we only had access to one temp sensor for this launch. While I have no interior temperature data at apex, I can tell you that when I opened up the package once it landed, the data logger was very warm. I think an interior temperature sensor is a good idea for future launches. Thanks for the interesting thought!



Wow, that's a great project, and you did an awesome job with the write-up! Very thorough and interesting! You've got my vote! :)

I do have a basic question on the launch side ... since your group has launched so many balloons, you guys must have learned a lot on the launch side as well, such as:

- Best place to buy the materials,
- Trade-offs of the different materials,
- Best places to launch from
- any laws that need to be observed ...

If you could recommend a place to start learning this information from, where would you point people to?

Thanks again for your write-up, you've got a bright future ahead of you!



Regarding the package we have tried a variety of things. We have used the typical styrofoam cooler, open packages, lunchboxes, etc. Each package has its advantages and disadvantages so it depends on your goals for the launch. When we launch 360 cameras, we use a totally open package. We laser cut a piece of whiteboard into a triangle shape. While this provides a clear line of sight, the payload is completely exposed to the harsh elements. Styrofoam boxes work but are often overkill. When we launch styrofoam coolers, they are typically half full. Packages need to stay under four pounds to meet regulations. My favorite package is a nice, insulated lunchbox. These provide warmth and protection like the styrofoam cooler yet with substantially less mass and weight. This Instructable is super helpful: <https://www.instructables.com/id/My-Space-Balloon-Project-Stratohab-Success-High/> Thanks for the vote and the nice comments!



Congratulations on a successful and fun project! You have a great future ahead of you. Just a note on wind chill values: Wind chill only applies to human skin, not instrument packages or anything else. The wind chill cooling effect is due to evaporation of the moisture in normal skin. Evaporation is greatly increased by wind and low humidity of the air. Also balloons are carried along by the winds so even though your balloon was in a 45 mph wind stream (as determined by the GPS measurements, I assume) there is zero relative wind on the balloon.



Very interesting. Thanks for the knowledge and support!



I'm not sure my first attempt posted...sorry if this is a duplicate.

I didn't see anything in the article about transmitters? How do you track the unit for recovery?
ps. Inspiring project, great work.



Thanks for the question. This unit logs the data on an attached MicroSD card. We use two SPOT GPS units to track our packages. Here is their website <https://www.findmespot.com/en/>

These units send us their location every 10 minutes over text and map the data automatically. If you are interested in building a system that communicates with the ground check out this Instructable: <https://www.instructables.com/id/Make-a-High-Altitude-Weather-Balloon-Data-Logger/>

Thanks for the support!



Thank you for the information...the Instructable project looks very fun



hey great project. And very well documented too but I still have lots of doubts and clarification that I need. I'm planning to launch one myself so I'd be very happy if you could share your email address or any other way I could contact you personally! Thanks in advance. :)



Send your questions to my Instructables inbox. I'll gladly answer them! Thanks for the support.



Hi, great instructable. You are just 15 and I think you have a very bright future ahead. Good luck!!



Thank you! I'm trying to learn as much as I can.



Nice! I have some balloons laying around but never got the motivation to buy gas and instruments.. I'm thinking about it again now!



Please do! It's a great opportunity. If you have any questions please reach out.



great



Thank you!



Wow! Fantastic!



Glad you enjoyed!



Thank you. I'm glad you enjoyed!