5-22-2020

Analysis of Low-Cost Computers for High Altitude Balloon Flight

Xiaoniu Du

Analysis of Low-Cost Computers for High-Altitude Balloon Flight

by

Xiaoniu Du

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

to

The Honors College

of

The University of Alabama in Huntsville

2020/5/22

Honors Capstone Director: Dr. Matthew W. Turner

Principal Research Engineer, The University of Alabama in Huntsville

Xiaoniu Du

_______2020/5/22___________

Student Date

05/22/2020

Director Date

Department Chair Date

William Wilkerson Digitally signed by William Wilkerson
Date: 2020.05.23 11:12:42 -05'00'

Honors College Dean Date

HONORS COLLEGE
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Honors College
Frank Franz Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

____Xiaoniu Du_________________________

Student Name (printed)

____Xiaoniu Du_________________________

Student Signature

__2020/5/22_________

Date

Analysis of Low-Cost Computers for High-Altitude Balloon Flight

Xiaoniu Du, Brady Richardson, Michael Angeles

Abstract

This project aimed at analyzing the effectiveness of two low-cost computers suitable for high-altitude balloon flight. The project team will consist of three team members, Xiaoniu Du, Michael Angeles, and Brady Richardson. A Raspberry Pi and a secondary low-cost computer Sipeed MAix BiT will be deployed to take measurements during the flight, and the performance of each analyzed and compared. The computers will be programmed so that pressure and temperature readings can be taken, as well as the latitude, longitude, and altitude of the balloon when these readings are taken. Both the Raspberry Pi and the low-cost computer were mounted on a pre-designed cone-shaped structure that was connected to the launching balloon. Before launch of the balloon three successful functional tests will be completed. These tests will ensure the sensors operate the entire length of the predicted flight time, as well as verify the GPS and altitude measurements. The construction of the balloon will be performed by Michael Angeles by use of a standard delta-frame kit provided by High Altitude Science, as well as flight predictions and execution. During the design and execution of the project, all FAA guidelines will be followed.

Introduction

As for development boards, Raspberry Pi has been heavily used in all kinds of control system at industry level, due to its high performance at a relatively small size. A Raspberry Pi as small as the size of a credit card is powerful enough to run a desktop set. Being aware of the fine ability of Raspherry Pi, we also want to compare its performance with another low-cost development board commonly used in AI controlling - Sipeed MAix BiT. For this purpose, the mission in this paper is to launch a balloon payload to 80,000 ft and meanwhile to record the temperature, pressure, altitude and GPS measurement. The two different computing boards pre-connected with two separate sensor systems will both be mounted on the balloon craft. After landing, the flight data collected by both systems will be compared and discussed.

Instruments

Item Name	Part Number
Raspberry Pi	
Sipeed Maixduino for RISC-V AI + IoT	102991150
Adafruit Precision NXP 9-DOF Breakout Board	FXOS8700 + FXAS21002
ELEGOO 17 Values 1% Resistor Kit Assortment	EL-CK-004
TUOFENG 22 awg Solid Wire-Solid Wire Kit	B07TX6BX47
Adafruit Ultimate GPS Breakout	PA6H1F1702
AUSTOR 100 Pcs PCB Board Kit	6.51355E+11
180degree Fisheye Lens 1080p Wide Angle Pc Web USB Camera. USB Camera Module for Android Windows. Cam Module Ir.	B00LQ854AG
Adafruit BMP280 I2C or SPI Barometric Pressure & Altitude Sensor	BMP280

Table 1. List of components for the payload

Table 1 shows the instruments used in this project, including the Raspberry Pi and Sipeed MAix as the computers for the two developed sensor systems, FXOS8700 3-Axis accelerometer and magnetometer, the FXAS21002 3-axis gyroscope, Adafruit Ultimate GPS Breakout as the latitude and longitude sensor, BMP280 as the pressure and altitude sensor, jumper wires, resistors, a USB camera, batteries and Printed Circuit Board (PCB). The budget breakdown is shown in the Appendix 1.

Procedure

```
import time
import board
import busio
import adafruit_fxos8700

i2c = busio.I2C(board.SCL, board.SDA)
sensor = adafruit_fxos8700.FXOS8700(i2c)

while True:
    accel_x, accel_y, accel_z = sensor.accelerometer
    mag_x, mag_y, mag_z = sensor.magnetometer
    print('Acceleration (m/s^2): ({0:0.3f}, {1:0.3f}, {2:0.3f})'.format(accel_x, accel_y, accel_z))
    print('Magnetometer (uTesla): ({0:0.3f}, {1:0.3f}, {2:0.3f})'.format(mag_x, mag_y, mag_z))
    time.sleep(1.0)
```

(a)

```
import time
import board
import busio
import adafruit_fxas21002c

i2c = busio.I2C(board.SCL, board.SDA)
sensor = adafruit_fxas21002c.FXAS21002C(i2c)

while True:
    gyro_x, gyro_y, gyro_z = sensor.gyroscope
    print('Gyroscope (radians/s): ({0:0.3f},  {1:0.3f},  {2:0.3f})'.format(gyro_x, gyro_y, gyro_z))
    time.sleep(1.0)
```

(b)

```python
# Simple GPS module demonstration.
# Will wait for a fix and print a message every second with the current location
# and other details.
import time
import board
import busio

import adafruit_gps

# Create a serial connection for the GPS connection using default speed and
# a slightly higher timeout (GPS modules typically update once a second).
# These are the defaults you should use for the GPS FeatherWing.
# For other boards set RX = GPS module TX, and TX = GPS module RX pins.
#uart = busio.UART(board.TX, board.RX, baudrate=9600, timeout=10)

# for a computer, use the pyserial library for uart access
import serial
uart = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=10)

# If using I2C, we'll create an I2C interface to talk to using default pins
#i2c = busio.I2C(board.SCL, board.SDA)

# Create a GPS module instance.
gps = adafruit_gps.GPS(uart, debug=False)     # Use UART/pyserial
#gps = adafruit_gps.GPS_GtopI2C(i2c, debug=False)  # Use I2C interface

# Initialize the GPS module by changing what data it sends and at what rate.
# These are NMEA extensions for PMTK_314_SET_NMEA_OUTPUT and
# PMTK_220_SET_NMEA_UPDATERATE but you can send anything from here to adjust
# the GPS module behavior:
#   https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf

# Turn on the basic GGA and RMC info (what you typically want)
gps.send_command(b'PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Turn on just minimum info (RMC only, location):
#gps.send_command(b'PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Turn off everything:
#gps.send_command(b'PMTK314,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Tuen on everything (not all of it is parsed!)
#gps.send_command(b'PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0')

# Set update rate to once a second (1hz) which is what you typically want.
gps.send_command(b'PMTK220,1000')
```

```python
# Or decrease to once every two seconds by doubling the millisecond value.
# Be sure to also increase your UART timeout above!
#gps.send_command(b'PMTK220,2000')
# You can also speed up the rate, but don't go too fast or else you can lose
# data during parsing.  This would be twice a second (2hz, 500ms delay):
#gps.send_command(b'PMTK220,500')

# Main loop runs forever printing the location, etc. every second.
last_print = time.monotonic()
while True:
    # Make sure to call gps.update() every loop iteration and at least twice
    # as fast as data comes from the GPS unit (usually every second).
    # This returns a bool that's true if it parsed new data (you can ignore it
    # though if you don't care and instead look at the has_fix property).
    gps.update()
    # Every second print out current location details if there's a fix.
    current = time.monotonic()
    if current - last_print >= 1.0:
        last_print = current
        if not gps.has_fix:
            # Try again if we don't have a fix yet.
            print('Waiting for fix...')
            continue
        # We have a fix! (gps.has_fix is true)
        # Print out details about the fix like location, date, etc.
        print('=' * 40)  # Print a separator line.
        print('Fix timestamp: {}/{}/{} {:02}:{:02}:{:02}'.format(
            gps.timestamp_utc.tm_mon,   # Grab parts of the time from the
            gps.timestamp_utc.tm_mday,  # struct_time object that holds
            gps.timestamp_utc.tm_year,  # the fix time.  Note you might
            gps.timestamp_utc.tm_hour,  # not get all data like year, day,
            gps.timestamp_utc.tm_min,   # month!
            gps.timestamp_utc.tm_sec))
        print('Latitude: {0:.6f} degrees'.format(gps.latitude))
        print('Longitude: {0:.6f} degrees'.format(gps.longitude))
        print('Fix quality: {}'.format(gps.fix_quality))
        # Some attributes beyond latitude, longitude and timestamp are optional
        # and might not be present.  Check if they're None before trying to use!
        if gps.satellites is not None:
            print('# satellites: {}'.format(gps.satellites))
        if gps.altitude_m is not None:
            print('Altitude: {} meters'.format(gps.altitude_m))
        if gps.speed_knots is not None:
            print('Speed: {} knots'.format(gps.speed_knots))
        if gps.track_angle_deg is not None:
            print('Track angle: {} degrees'.format(gps.track_angle_deg))
        if gps.horizontal_dilution is not None:
            print('Horizontal dilution: {}'.format(gps.horizontal_dilution))
        if gps.height_geoid is not None:
            print('Height geo ID: {} meters'.format(gps.height_geoid))
```

(c)

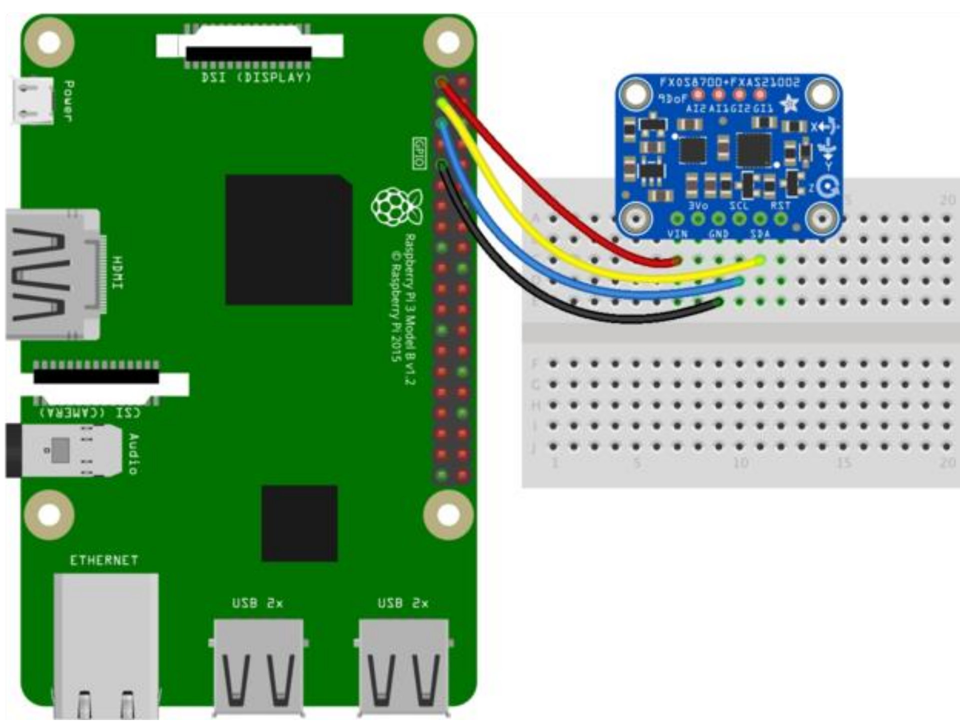
```python
import board
import busio
import adafruit_bmp280
i2c = busio.I2C(board.SCL, board.SDA)
sensor = adafruit_bmp280.Adafruit_BMP280_I2C(i2c)

print('Temperature: {} degrees C'.format(sensor.temperature))
print('Pressure: {}hPa'.format(sensor.pressure))
```

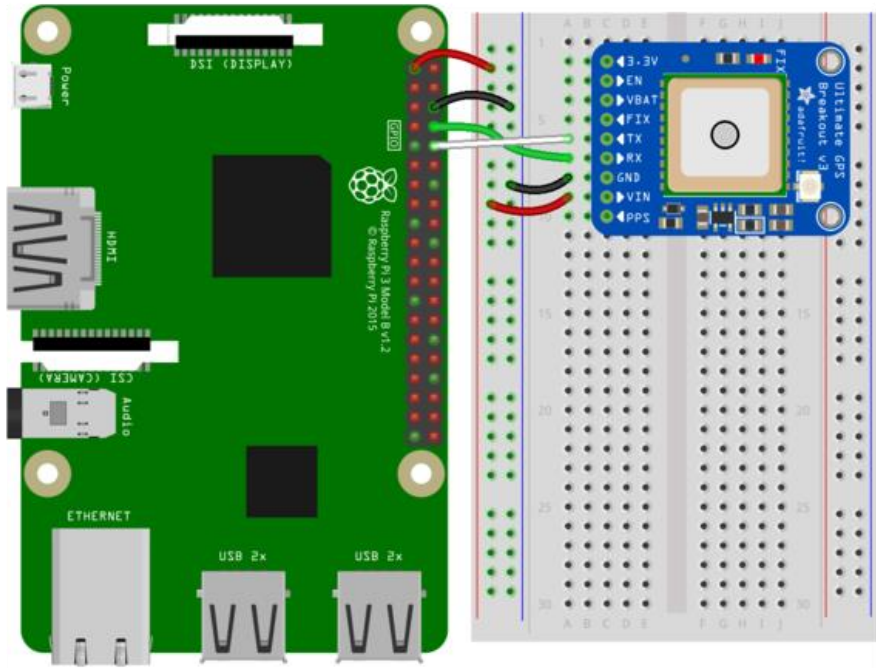
(d)

Figure 1. Programming code for (a) FXOS8700; (b) FXAS21002C; (c) Adafruit Ultimate GPS Breakout; (d) BMP280 pressure and altitude sensor.

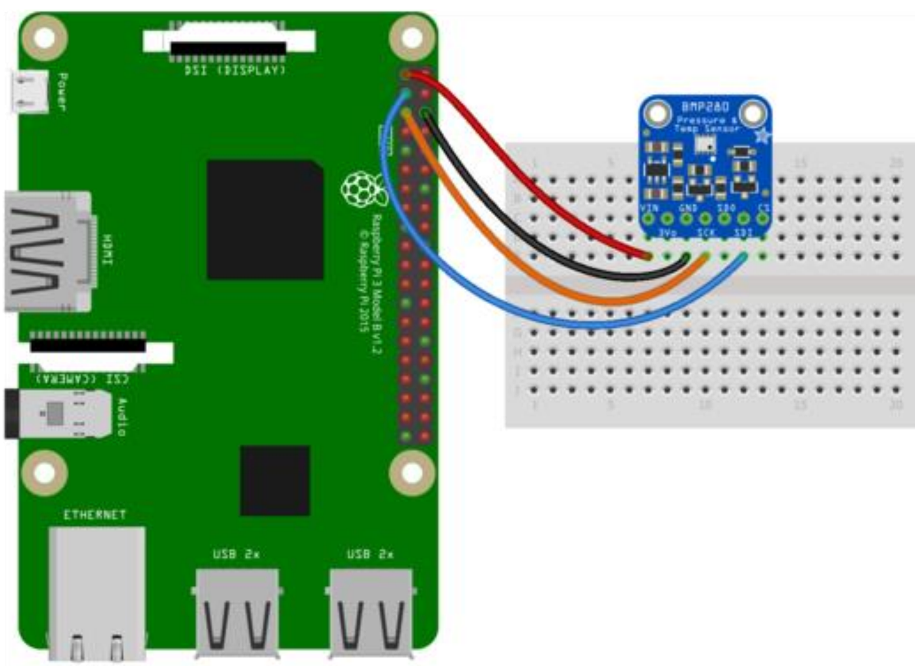
Figure 1 shows the programming code for the motion sensor, GPS breakout and pressure and altitude sensor used in the project. The coding was done on CircuitPythoon for both computer systems. After programming, the sensor systems are soldered to the PCB together with the computers and the batteries. Figure 2 shows the connecting configuration for the sensors.



(a)


DSI (DISPLAY)
Power
HDMI
CSI (CAMERA)
Audio
ETHERNET
USB 2x
USB 2x
GPIO
Raspberry Pi 3 Model B v1.2
© Raspberry Pi 2015
3.3V
EN
VBAT
FIX
TX
RX
GND
VIN
PPS
FIX
Ultimate GPS
Breakout v3
adafruit!

(b)


DSI (DISPLAY)
Power
HDMI
CSI (CAMERA)
Audio
ETHERNET
USB 2x
USB 2x
Raspberry Pi 3 Model B v1.2
© Raspberry Pi 2015
BMP280
Pressure &
Temp Sensor
VIN
GND
SDO
CS
3Vo
SCK
SDI

(c)

Figure 2. Circuit connection schematic for (a) FXOS8700 + FXAS21002C motion sensor; (b) Adafruit Ultimate GPS Breakout; (c) BMP280 pressure and altitude sensor.

The two developed sensor systems are mounted to a pre-designed cone-shaped structure made by wood. The larger ring on the bottom provide firmer attachment with the balloon and more stability during the flight. The balloon-carried payload is launched to a minimum altitude of 80,000 ft, and the sensors are programmed to record the temperature, pressure, altitude and GPS data throughout the entire process. Three successful functional tests for the sensor systems are performed before the launch. During the functional test, the sensors, computers and batteries are set to start working as they should during the actual launch at three different locations, where the actual temperature, pressure, altitude and GPS of each specific location are preemptively measured. Then, the data collected by the sensor systems are compared with the actual measured data. If the data show agreement, the functional test is successful; if not, we adjust the component where the data show disagreement with the true data and run the test again until it succeeds. The SD (TF) card which recorded the flight data will be restored after the landing piece of the balloon craft is collected. After landing, the data are extracted and compared between the two sensor systems powered by Raspberry Pi and Sipeed Maixduino respectively and the results are discussed. The launching operations follow the FAA guidelines [1]. Figure 3 shows our tentative schedule of every stage before and after the flight.

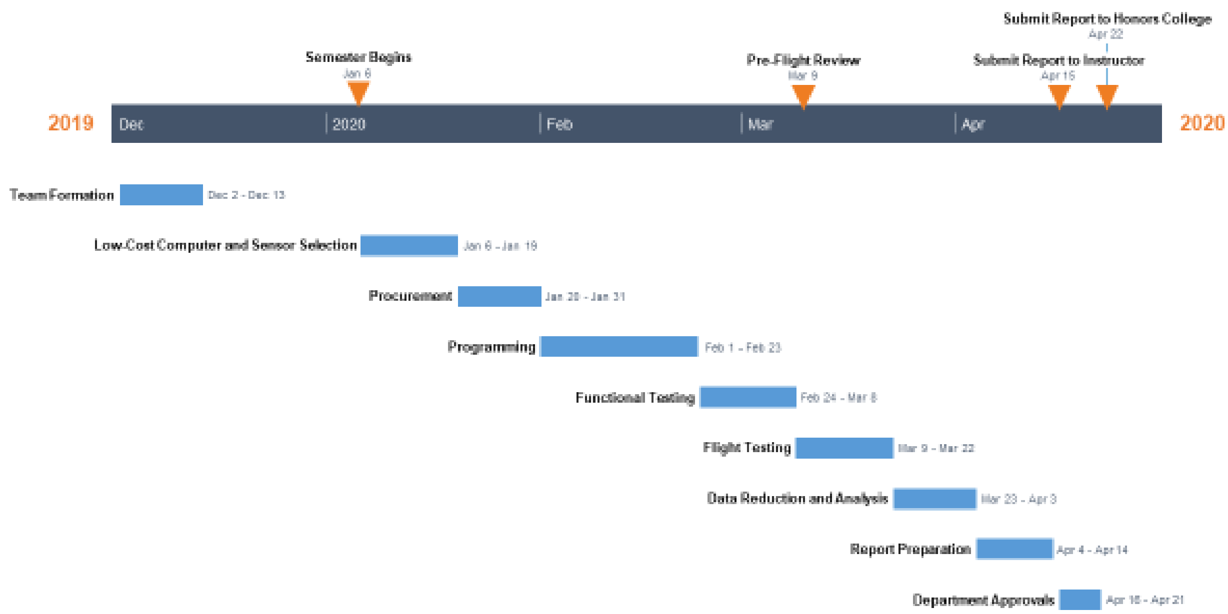


Figure 3. Tentative Schedule before and after the launch of the balloon payload

Results

The project started from the beginning of January 2020 and is supposed to be carried on till the end of April. However, because of the quarantine caused by the coronavirus pandemic, we only managed to proceed to the programming stage. The programming codes for the FXOS8700 & FXAS21002C motion sensor and BMP280 pressure and altitude sensor worked effectively as they can display the data which showed agreement to the actual situation when we were testing the code. In this case, the velocity showed zero (because the sensor was stationary when we were testing the code) and the pressure and altitude showed agreement to the real-world situation. The code for Adafruit Ultimate GPS Breakout was more difficult to program. It started showing values, but further calibration needs to be done.

	Value	Units
Payload Mass	1.81	kg
Lift	5.44	kg
Gross Lift	6.943108	kg
He Volume	6.764525	cubic meter
Ascent Rate	7.39	meters/sec
Burst Altitude	99148.5	ft
Time to Burst	68.1483	min
Temp at Launch	80	F
Pressure at launch	14.7	Psia
Balloon Mass	1.5	kg
Balloon Burst Diameter	9.44	meter
Decent Rate	49.44	meters/sec
Maximum Altitude	100,000	ft

Table 2. Result of the calculations by HAB Burst Model and Decent Model

	Value	Units
Launch Date	2020/4/25	
Launch Time	8:00	AM
Launching Latitude	34.72415	
Launching Longitude	-86.644	
Landing Latitude	34.7349	
Landing Longitude	-86.6381	

Table 3. Latitude and longitude prediction of the landing location

The balloon burst, decent rate and trajectory are calculated by mathematical models and the results are shown in the Table 2 [2] [3]. The Maximum altitude is set to be 100,000 ft due to the minimum altitude of 80,000 and the burst altitude of 99148.5 ft. The balloon is estimated to be 1500 kg and have a burst diameter of 9.44 m according to the Totex Balloon Data. The decent rate is calculated to be 49.44 meters/sec given the total mass and the balloon diameter. Based on the modeled launch date of April 25th at 8:00 AM, and the modeled launch location of the campus of The University of Alabama in Huntsville, Table 3 shows the trajectory prediction that the balloon payload would be landing at the latitude of 34.7349 and longitude of -86.6381.

Appendix

Item Name	Part Number	Price	Quantity	SubTotal
Adafruit BMP280 I2C or SPI Barometric Pressure & Altitude Sensor	BMP280	$9.95	2	$19.90
Adafruit Precision NXP 9-DOF Breakout Board	FXOS8700 + FXAS21002	$14.95	2	$29.90
ELEGOO 17 Values 1% Resistor Kit Assortment	EL-CK-004	$10.86	1	$10.86
TUOFENG 22 awg Solid Wire-Solid Wire Kit	B07TX6BX47	$15.99	1	$15.99
Adafruit Ultimate GPS Breakout	PA6H1F1702	$39.95	2	$79.90
AUSTOR 100 Pcs PCB Board Kit	6.51355E+11	$15.99	1	$15.99
Sipeed Maixduino for RISC-V AI + IoT	102991150	$12.90	2	$25.80
180degree Fisheye Lens 1080p Wide Angle Pc Web USB Camera.USB Camera Module for Android Windows .Cam Module Ir.	B00LQ854AG	$45.00	2	$90.00
TOTAL				$288.34

References

[1] Federal Aviation Administration, Balloon Flying Handbook (FAA-H-8083-11A), June 14, 2013.

[2] Model Rocket Parachute Descent Rate Calculator, https://descentratecalculator.onlinetesting.net/

[3] Trajectory Prediction Model, http://predict.habhub.org/