EMANUEL BOMBASARO

TITAN 1

DESIGN AND MISSION DOCUMENTATION

contact via e-mail at titan.hab@gmail.com.

First published, September 2015

Titan

1. Greek Mythology any of the older gods who preceded the Olympians and were the children of Uranus (Heaven) and Gaia (Earth). Led by Cronus, they overthrew Uranus; Cronus' son, Zeus, then rebelled against his father and eventually defeated the Titans.
   (as nouna titan) a person or thing of very great strength, intellect, or importance: a titan of American industry.

2. Astronomy the largest satellite of Saturn (diameter 5150 km), the fifteenth closest to the planet, discovered by C. Huygens in 1655. It is unique in having a hazy atmosphere of nitrogen, methane, and oily hydrocarbons.

Contents

1 Preliminaries

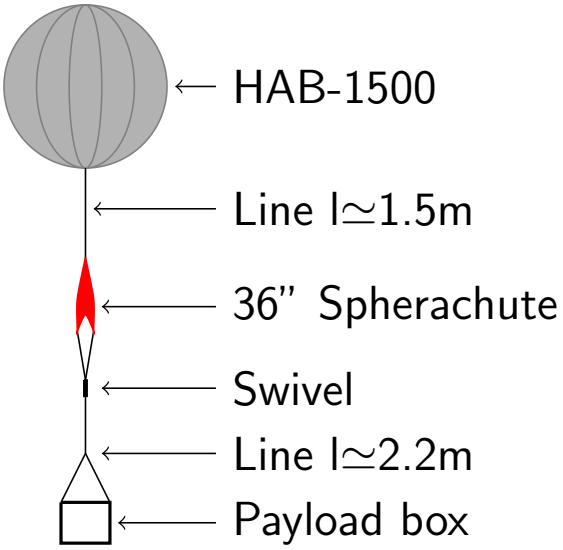
This report outlines the process of developing and construction of a high altitude balloon (HAB) reaching a peak altitude of 35 000 m. A mission is completed with success when the payload box reaches at least 35 000 m and is recovered, logged sensor data and captured images extracted with success.

After the mission a Flight Data Report will be prepared, containing all recorded flight data and a discussion of those. Further, a review of the individual components and their operation will be stated.

The idea of constructing a HAB came while reading following two papers on the Huygens probe, J.-P. Lebreton and D. L. Matson *The Huygens Probe: Science, Payload And Mission Overview*, Space Science Reviews 104: 59100, 2002 and R. D. Lorenz *Spin of Planetary Probes in Atmospheric Flight*, Journal of the British Interplanetary Society, vol 59, No.8, 273-282, August 2006. Huygens landed on Saturn's moon Titan January 14th 2004. Reading about the challenges faced during the Cassini-Huygens mission (http://www.esa.int/Our_Activities/Space_Science/Cassini-Huygens/ made the author curious in exploring *at least* the earth atmosphere.

1.1 Technical Characteristics

Bellow the main technical characteristics of the Titan HAB are stated together with a schematic view of the balloon. This is followed by a description of the used sensors and image capture device.



Peak Altitude: 35 104 m

Ascent Velocity: 5.59 m s^{-1}

Descent Velocity: 4.78 m s^{-1}

Ascent Time: 105 min

Descent Time: 122 min

Total Flight Time: 227 min

Payload Weight: 0.720 kg

Length over all: ≃7 m

Balloon Volume at Launch: 3.460 m^3

Balloon Lift at Launch: 3.548 kg

Area Weight: 5.06 g cm^{-2}

An Arduino Mega 2560 is the main micro controller used as flight computer. It is based on the ATmega2560 microcontroller. The global position of the HAB is identified with a MAX-M8 series GPS module and the data are transmitted via a Radiometrix MTX2 434 MHz Radio Module, which is a frequency agile 434 MHz 10 mW radio module.

Following sensors and modules are attached to the Arduino Mega 2560:

MTX2 Radiometrix MTX2 434 MHz Radio Module.

HX1 VHF Narrow Band FM 300 mW Transmitter, 144.800 MHz, used for APRS.

MAX-M8 GPS module used for position (longitude, latitude and altitude) and time acquisition.

DS18B20 Temperature sensor on HABuino showing the temperature of the flight computer compartment. This temperature should remain most near to 20 °C. Any temperature variation will effect the transmission frequency of the radio module.

MCP9808 Maximum accuracy digital temperature sensor measuring air temperature.

HTU21DF Temperature and humidity sensor measuring air temperature and relative humidity of the air.

MPL3115A2 Precision altimeter mainly used for measuring atmospheric pressure, but also temperature and altitude is detected.

TSL2561 Light to digital converter

BST-BMP180 Pressure sensor mainly used for measuring atmospheric pressure, but also temperature and altitude is detected.

L3GD20 3D gyroscope

LSM303DLHC 3D accelerometer and 3D magnetometer module

LSH20 Saft LSH 20 battery used as power supply with 3.6 V and 13.0 A h. The power feeds into the low input voltage synchronous boost converter TPS61201 on the HABuino shield.

Further details on the sensors can be found in Tab. 1 and in section 2. The wiring is explained in section 2.1.

The image capturing is done independent from the flight computer with a GoPro HERO4 Silver camcorder, see section 4. Basically an image is captured every second.

1.2 Measured Quantities

Tab. 1 gives an overview of measured quantities and the corresponding variable names. The variable names correspond to those used in the fight computer software.

Component	Quantity	Unit, Precision	Description
MAX-M8	*time*	hh:mm:ss	
	lon	°, -	Horizontal position accuracy around
	lat	°, -	2.5 m in pedestrian mode.
	alt	m, -	
	sats	1, -	Number of satellites used for position determination. We want to have a minimum of 4 satellites.
DS18B20	*temperature1*	°C, ±0.5	Temperature operating range −55 °C to 125 °C.
MCP9808	*temperature5*	°C, ±0.5	Temperature operating range −55 °C to 125 °C.
HTU21DF	*temperature3*	°C, ±0.5	Temperature operating range −40 °C to 125 °C.
	humidity1	%RH, ±2	Humidity operating range 0 %RH to 100 %RH.
MPL3115A2	*pressure1*	Pa, $\pm 1.5 \times 10^3$	Pressure operating range 20×10^3 Pa to 110×10^3 Pa.
	alt1	m, ±0.3	$alt1 = 44330.77 \left[1 - (p/p_0)^{0.1902632}\right] + OFF_H(Register\ Value)$ with p_0 sea level pressure 101 326 Pa.
	temperature4	°C, ±3	Temperature operating range −40 °C to 85 °C.
TSL2561	*lum1*	-	
	ir1	-	
	full1	-	
	lux1	lx, ?	
BST-BMP180	*pressure2*	Pa, -0.1×10^3	Pressure operating range 30×10^3 Pa to 110×10^3 Pa.
	alt2	m, -	$alt2 = 44330 \left[1 - (p/p_0)^{1/5.255}\right]$ with p_0 sea level pressure 101 326 Pa.
	temperature6	°C, ±1	Temperature operating range −40 °C to 85 °C.
LSM303DLHC	*accX*	$\mathrm{m\,s^{-2}}$,	
	accY	$\mathrm{m\,s^{-2}}$,	
	accZ	$\mathrm{m\,s^{-2}}$,	
	magX	G, ±3	
	magY	G, ±3	
	magZ	G, ±3	
L3GD20	*gyroX*	$^\circ\,\mathrm{s^{-1}}$, 70×10^{-3}	
	gyroY	$^\circ\,\mathrm{s^{-1}}$, 70×10^{-3}	
	gyroZ	$^\circ\,\mathrm{s^{-1}}$, 70×10^{-3}	

Component	Quantity	Unit, Precision	Description
TPS61201	*battvaverage*	mV, ±5	Measured through Arduino analog pin A0. Operating input voltage range from 0.3 V to 5.5 V. 600 mA output current at 5 V ($VIN \geq$3 V) and 300 mA at 3.3 V ($VIN \geq$2.4 V).

Table 1: Technical specification of components and measured quantities with variable names.

1.3 Communicated Data

The transmission is done with RTTY with carrier shift of 425 (can be freely chosen) a Baud rate of 50 (can be freely chosen) using 7-bit (selectable also 8-bit) ASCII and 2 stop bits (selectable also 1 bit). The values selected are the recommended values. The transmission frequency is 434.485 MHz. The transmission frequency can be chosen within the interval 433.875 MHz to 434.650 MHz.

Following string is communicated with the transmission module:
callsign, count, hour:minute:second, lat, lon, maxalt, sats, temperature1, battvaverage, errorstatus, checksum
An example is shown in List. 1 which are transmitted and received data.

Listing 1: An example of some lines of a transmitted and received data.

```
1 RX 434486878 : RTTY (2015-08-20 16:23Z): $$$$$TITAN_1
    ,16,16:23:53,55.747342,11.572041,5,6,29,3414,8*41F6
2 RX 434486881 : RTTY (2015-08-20 16:24Z): $$$$$TITAN_1
    ,17,16:24:05,56.747345,11.572083,18,6,29,3415,8*9805
3 RX 434486884 : RTTY (2015-08-20 16:24Z): $$$$$TITAN_1
    ,18,16:24:17,57.747379,11.572097,13,7,29,3416,8*BBAE
4 RX 434486886 : RTTY (2015-08-20 16:24Z): $$$$$TITAN_1
    ,19,16:24:33,58.747334,11.572081,13,7,29,3415,8*E0A5
```

The checksum is calculated using the Xmodem-CRC protocol, see http://www.atmel.com/webdoc/AVRLibcReferenceManual/group__util__crc_1gaca726c22a1900f9bad52594c8846115f.html.

Specification about the receiver is given in section 5.

1.3.1 APRS

During development it was considered to use the HX1 module on the HABduino to to transmit position based on the APRS standard. However, for this mission it was not considered further. It was used for development purpose and will find its use in a consecutive mission.

1.4 Logged Data

Before every transmission all data measured by the sensors, including position data from the GPS module are saved to an on-board micro SD memory card. Thus data are logged approximately every 14 s.

The function `writeToSD()` (Line 597 in List. 3) appends the string containing the current sensor data to the file `logfilename`. The resulting log file is a delimiter file with a `,` used as delimiter. The data for both the transmitted and logged variables is prepared in the function `prepare_data()` (Line 408 in List. 3).

Tab. 2 shows the individual variables written to the log file with corresponding unit and format. The variables are written to the file exactly in the order as presented in Tab. 2 starting from top. An example of some lines of a logged file are shown in List. 2.

Variable name	**Variable symbol**	**Unit**	**Format**	**Example**
callsign		-	9 character string	TITAN_1
count		1	integer	108
hour:minute:second		hh : mm : ss	int:int:int	14:18:32
lat		°	float	56.747320
lon		°	float	11.572087
maxalt		m	32-bit integer	20
sats		1	integer	6
temperature1		°C	integer	23
battvaverage		mV	integer	169
errorstatus		-	16-bit integer	8
lum1		-	32-bit integer	2424882
ir1		1	integer	37
full1		1	integer	50
lux1		lx	integer	2
temperature3		°C	float	20.33
humidity1		%RH	float	44.65
pressure1		Pa	float	102427
alt1		m	float	-92.80
temperature4		°C	float	18.06
temperature5		°C	float	19.94
pressure2		Pa	float	102519
alt2		m	float	-98.86
temperature6		°C	float	21.60
gyroX	Ω_x	$^\circ\,s^{-1}$	integer	-2
gyroY	Ω_y	$^\circ\,s^{-1}$	integer	-0
gyroZ	Ω_z	$^\circ\,s^{-1}$	integer	-0
accX	a_x	$m\,s^{-2}$	float	0.32
accY	a_y	$m\,s^{-2}$	float	-0.08

Variable name	Variable symbol	Unit	Format	Example
accZ	a_z	$\mathrm{m\,s^{-2}}$	float	10.26
magX	m_x	µT	float	-0.36
magY	m_y	µT	float	0.24
magZ	m_z	µT	float	-0.17
roll	γ	°	float	-0.44
pitch	ρ	°	float	-1.80
head	ψ	°, 0° North	float	146.17

Table 2: Logged variables with units and format specification. Value of example is first line in log file shown in List. 2.

Listing 2: An example of some lines of a log file.

```
1 TITAN_1,108,14:18:32,56.747320,11.572087,20,6,23,169,8,2424882,
    37,50,2,20.33,44.65,102427,-92.80,18.06,19.94,102519,-98.86,21.60,
    -2,-0,-0,0.32,-0.08,10.26,-0.36,0.24,-0.17,-0.44,-1.80,146.17
2 TITAN_1,109,14:18:47,57.747348,11.572043,19,6,23,168,8,2490418,
    38,50,2,20.33,44.66,102431,-88.92,17.94,19.87,102521,-99.02,21.60,
    -2,-0,-0,0.30,-0.05,10.25,-0.36,0.24,-0.17,-0.31,-1.67,146.32
3 TITAN_1,110,14:18:59,58.747313,11.572031,8,6,23,168,8,2490418,
    38,50,2,20.32,44.67,102449,-91.56,18.06,19.87,102523,-99.19,21.60,
    -2,-0,-0,0.31,0.00,10.20,-0.36,0.24,-0.17,0.00,-1.76,146.16
4 TITAN_1,111,14:19:11,59.747360,11.572076,16,6,23,168,8,2359345,
    36,49,2,20.33,44.69,102432,-90.88,18.06,19.94,102515,-98.53,21.60,
    -2,-0,-0,0.19,0.09,10.14,-0.36,0.24,-0.17,0.49,-1.06,146.02
5 TITAN_1,112,14:19:27,53.747305,11.572081,18,7,23,168,8,2424882,
    37,50,2,20.32,44.69,102427,-89.93,18.00,19.94,102520,-98.94,21.60,
    -2,-0,-0,0.31,0.00,10.24,-0.36,0.24,-0.17,0.00,-1.76,145.91
6 TITAN_1,113,14:19:37,52.747332,11.572083,25,6,23,168,8,2424882,
    37,50,2,20.33,44.70,102445,-90.36,18.06,19.87,102520,-98.94,21.60,
    -2,-0,-0,0.34,-0.05,10.29,-0.36,0.24,-0.17,-0.31,-1.88,146.32
7 TITAN_1,114,14:19:54,51.747262,11.572100,28,6,23,168,8,4522072,
    69,88,2,20.34,44.71,102429,-91.87,18.12,19.87,102512,-98.28,21.60,
    -2,-0,-0,0.35,-0.39,10.28,-0.36,0.29,-0.15,-2.19,-1.97,141.29
```

2 Flight Computer Hardware

Following section gives some more details about the used hardware components and how they individual parts are connected together.

2.1 Wiring and General Circuit Schematic

The general outline of the connection is visualised in Fig. 1. The connections are based on the wiring needs for the individual sensors. The power supply is connected trough the HABuino shield.

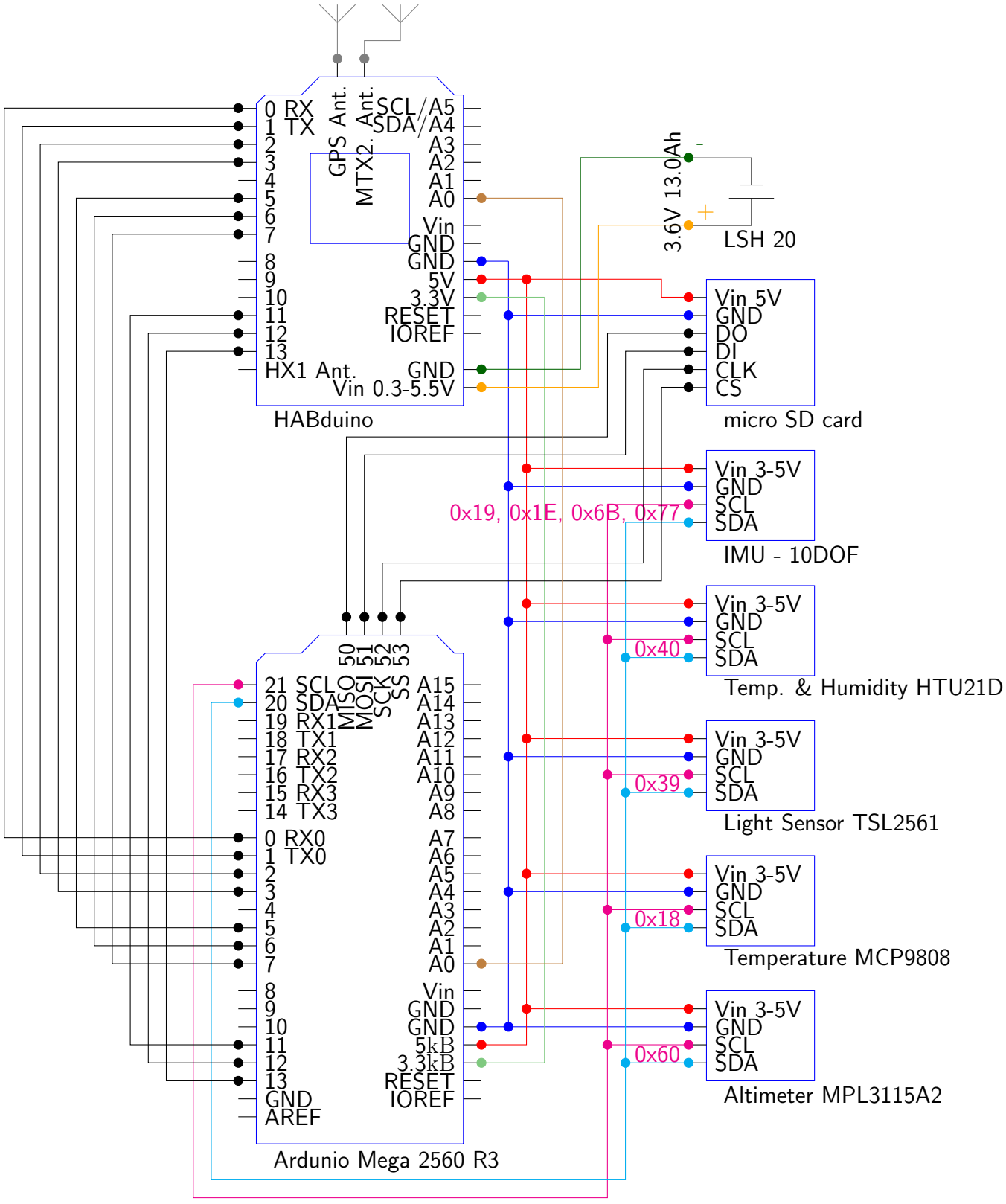


Figure 1: General circuit schematic

On top of the Arduino Mega 2560 a prototype shield is mounted. The prototype shield is only connecting towards the Arduino Mega 2560 pins which are actually used by the

individual components, see in Fig. 2 the prototype shield between the Arduino Mega 2560 R3 (bottom) and the HABduino shield (top). This is done to not increasing the weight of the flight computer uselessly.

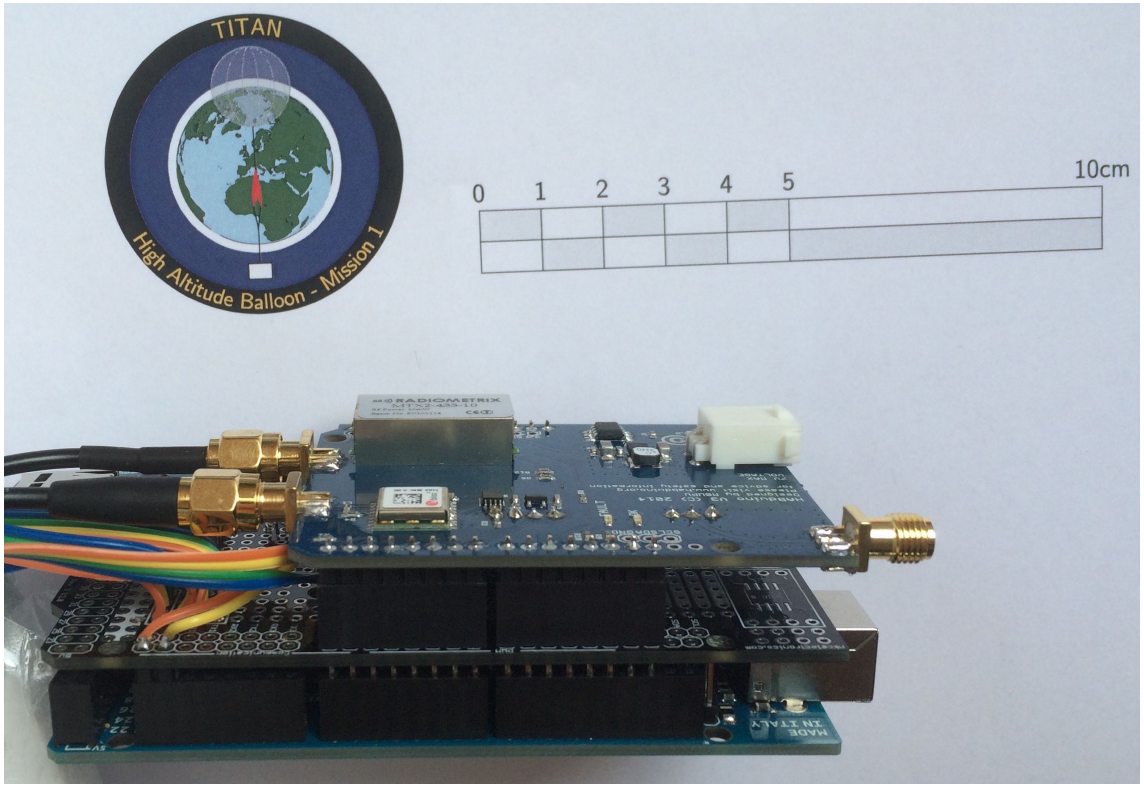


Figure 2: Flight computer, laterale view.

The positive bus is set to 5 V and the reset switch is installed in case the Arduino has to be rested manually. Stacking female headers are installed form pins Aref up to RX (upper row from right to left in Fig. 2) and pins Reset up to A5 (lower row from right to left in Fig. 2). Further, pin 20 (SDA) and 21 (SCL) as well pins 50 (MISO), 51 (MOSI), 52 (SCK) and 53 (SS) for SPI communication are connected to the Arduino Mega 2560 with male headers. The sensors are connected via cable to the prototype board where the SD-card break out board is directly soldered onto the prototype board using male headers.

Fig. 3 shows the assembled flight computer with the HABduino stacked on top. Cables to antenna and sensors visible on the left. Further the micro-SD card breakout board is seen mounted on top of the prototype shield.

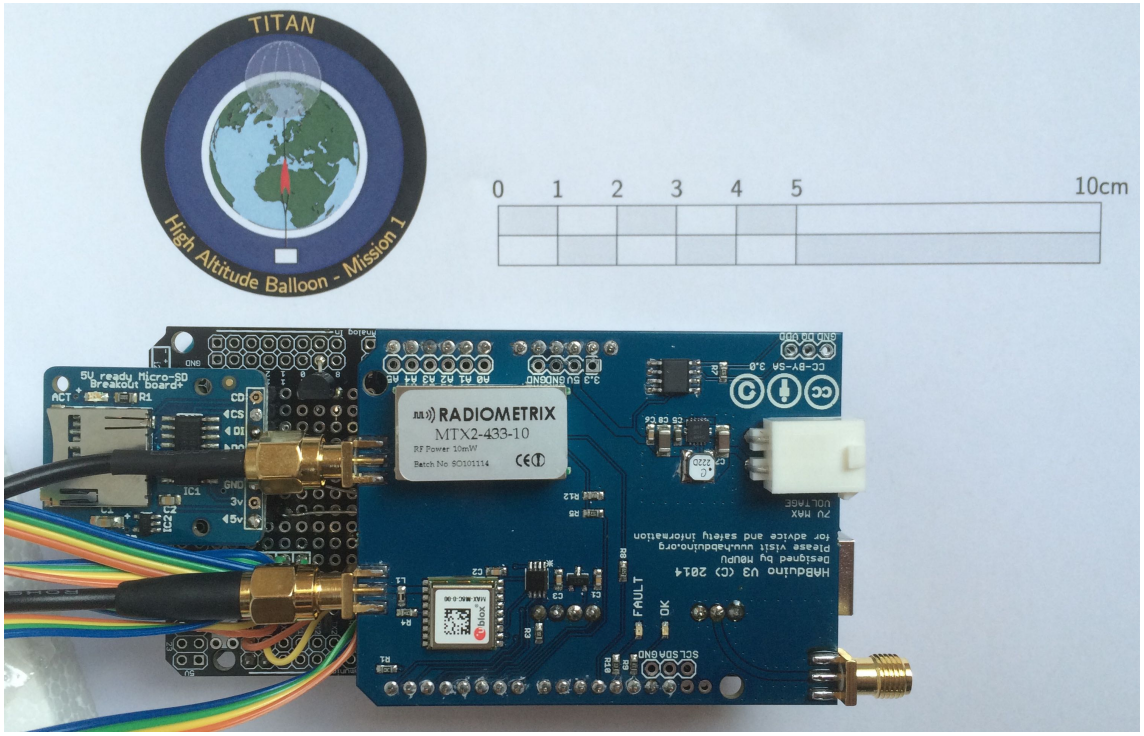


Figure 3: Flight computer, top view.

2.2 Microcontroller

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.

2.2.1 Technical Details

Element	Specifications
Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 kB of which 8 kB used by bootloader
SRAM	8 kB
EEPROM	4 kB
Clock Speed	16 MHz

Length	101.98 mm
Width	53.63 mm
Height	15.29 mm
Weight	34.9g

Table 3: Technical specification Arduino Mega 2560 R3

2.2.2 Power

The Arduino Mega can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector. The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

VIN The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

5V This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.

3.3V A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

GND Ground pins.

IOREF This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

2.2.3 Memory

The ATmega2560 has 256 kB of flash memory for storing code (of which 8 kB is used for the bootloader), 8 kB of SRAM and 4 kB of EEPROM (which can be read and written with the EEPROM library).

2.2.4 Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using `pinMode()`, `digitalWrite()`, and `digitalRead()` functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

Serial 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX). Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.

External Interrupts 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2). These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.

PWM 2 to 13 and 44 to 46 Provide 8-bit PWM output with the analogWrite() function.

SPI 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.

LED 13 There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

TWI 20 (SDA) and 21 (SCL) Support TWI communication using the Wire library.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function. There are a couple of other pins on the board:

AREF Reference voltage for the analog inputs. Used with analogReference().

Reset Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

2.2.5 Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5 V) serial communication. An ATmega16U2 on the board channels one of these over USB and provides a virtual com port to software on the computer

(Windows machines will need a `.inf` file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2/ATmega16U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).
A SoftwareSerial library allows for serial communication on any of the Mega2560's digital pins. The ATmega2560 also supports TWI and SPI communication. The Arduino software includes a Wire library to simplify use of the TWI bus; see the documentation for details. For SPI communication, use the SPI library. http://arduino.cc/en/Main/ArduinoBoardMega2560

2.3 Tracking

The HABduino board is a GPS receiver and radio transmitter shield designed for tracking high altitude balloon flights utilising an Arduino Uno/Duemilanove. For this application the included software was modified to use the shield on an Arduino Mega 2560. For more information see http://www.habduino.org.

The HABduino Kit Comprises of :

- Habduino Shield.
- GPS Antenna with 0.5 m cable.
- RG174 Pig Tail to make a suitable antenna (See this guide http://ukhas.org.uk/guides:payloadantenna)
- Battery holder + clip

Features:

- Compatible with Duemilanovo and Uno boards (Leonardo not supported)
- On board step up to permit operation from 2 x AA batteries.
- Ublox MAX M8C GPS suitable for high altitude use.
- 70cms frequency agile TCXO equipped transmitter for RTTY.
- Optional 300 mW 2 meter transmitter for APRS
- Interrupt driven simultaneous APRS and RTTY transmission.
- On board temperature sensor.

The open-source software provides these features:

- Radio telemetry with GPS and sensor data using UKHAS format.
- Supplied as github repository with instructions.
- Open hardware design.
- Optional APRS transmission alongside RTTY
- For latest news, code and documentation please visit: https://github.com/habduino

2.3.1 Transmitter, MTX2

The Radiometrix MTX2 434 MHz Radio Module is a frequency agile 434 MHz 10 mW radio module. Similar to the NTX2B-FA but much smaller. The MTX2 can be used in any circuit the original NTX2/NTX2B was used in (although the pin spacing is different). This unit features a TCXO for extra frequency stability plus frequency agility. The MTX2 allows selection of channels via a serial connection to the EN pin, additionally four fixed channels can also be selected via the parallel pins. The EN pin can still be used to turn the device on and off.

This module is suitable for license exempt (in Europe) giving out a maximum power of 10 mW. The module family is well proven in the HAB field recording reception distances in excess of 500 km.

Radiometrix are understandably unable to offer technical support on these modules when used in high altitude ballooning therefore please direct all technical support queries regarding the operation of these modules to the UKHAS mailing list or the #highaltitude IRC channel.

Dimensions 23 x 12.5 x 7 mm

Transmit power 10 dBm (10 mW)

Supply 2.9 V to 15 V at 18 mA

Recommended PCB Hole 1 mm

Weight 3.4 g

Link: http://www.radiometrix.com/conten/mtx2
Guide on linking the NTX2 to an Arduino here http://ukhas.org.uk/guides:linkingarduinotontx2.
Range statistics http://ukhas.org.uk/general:uk_records#radio_range_10mw_50_baud.

2.3.2 Transmitter, HX1

The special HX1 transmitter modules offer a 300 mW RF output VHF data link in Radiometrix SIL standard pin-out and footprint. This makes the HX1 ideally suited to those low power applications where existing narrow band and wideband transmitters provide insufficient range. Together with the matching NRX1 or BiM1R receiver a one-way radio data link can be achieved over a distance up to 10 km with suitable choice of data rate and antennas.

- Standard frequencies: 144.390, 144.800 and 169.4125 MHz
- Data rates up to 3 kbps
- Usable range over 10 km
- Fully screened
- Low power requirements

The HX1 is a narrow band radio transmitter module for use in long range data transfer applications at ranges up to 10 km. HX1 transmitter circuit is the BiM1T transmitter circuit in the TX1 pin-out with slightly enlarged dimension to accommodate extra Power Amplifier circuit to produce 300 mW RF output and available for operation on 169.4 to 169.475 MHz European licence exempt frequency band. Applications:

- Asset Tracking and Tracing
- Meter reading systems
- Automatic Position Reporting System (APRS)

Transmit power 300 mW (24.7 dBm)

Operating frequency 144.800 MHz

Channel spacing 25 kHz

Supply 5 V (regulated)

Current consumption 140 mA nominal transmit

Data bit rate 3 kbps or 10 kbps max.

Dimensions 43 x 15 x 5 mm

The HX 1 transmitter is mounted on the HABduino module, but not further used as APRS is not used for the first Titan HAB mission. http://www.radiometrix.com/content/hx1

2.4 Transmission Antennas

The length L in m of a 1/4 wave antenna can be calculated as follows.

$$L = 0.95 \frac{1}{4} \frac{c}{f} \tag{1}$$

with, c speed of light 299 792 458 m s^{-1}, f transmission frequency in Hz. Thus, the length L of the antenna is 164 mm. According to the MTX2 and HX1 transmitter data sheet this is the antenna with best range ratings. Additionally four radials were added, see Fig. 4

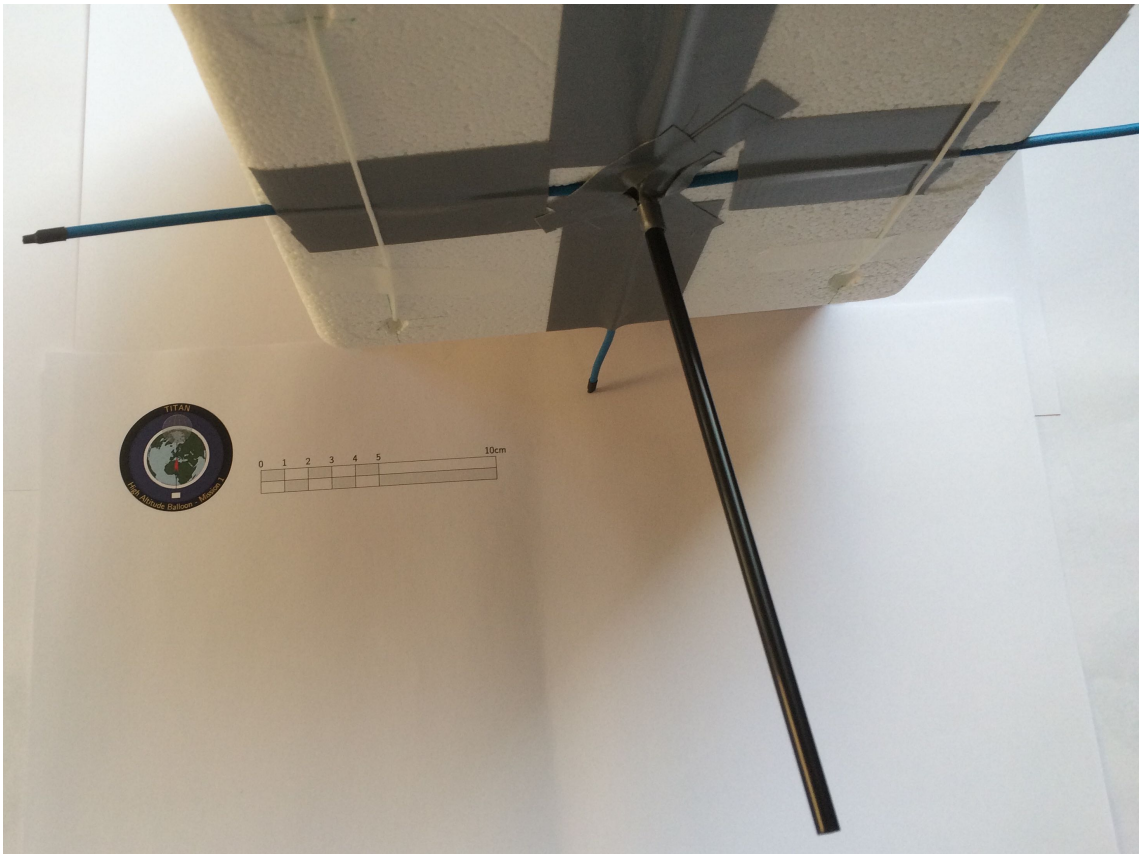


Figure 4: Transmission antenna, 1/4 wave @ 434.485 MHz with antenna length L 164 mm mounted on the bottom of the payload box.

The antenna is mounted on the bottom of the payload box pointing towards the ground, see Fig. 4.

Guide on linking the NTX2 to an Arduino here http://ukhas.org.uk/guides:linkingarduinotontx2.
Range statistics http://ukhas.org.uk/general:uk_records#radio_range_10mw_50_baud.

2.5 GPS Module

The MAX-M8 series of standalone concurrent GNSS modules is built on the exceptional performance of the u-blox M8 engine in the industry proven MAX form factor. With dual-frequency RF front-end, the u-blox M8 concurrent GNSS engine is able to intelligently use the highest amount of visible satellites from two GNSS (GPS, GLONASS and BeiDOU) systems for more reliable positioning.

The MAX-M8 series provides high sensitivity and minimal acquisition times while maintaining low system power. The MAX-M8C is optimized for cost sensitive applications, while MAX-M8Q/W provide best performance and easier RF integration. http://www.u-blox.com/en/gps-modules/pvt-modules/max-m8-series-concurrent-gnss-modules.html

- u-bloxs smallest LCC package
- Concurrent reception of GPS/QZSS, GLONASS, BeiDou
- Industry leading −167 dB m navigation sensitivity
- u-blox AssistNow GNSS Online, Offline and Autonomous
- Product variants to meet performance and cost requirements
- Pin-to-pin and software compatible with MAX-7 and MAX-6

2.5.1 Features

- Receiver type
  - 72-channel u-blox M8 engine
  - GPS/QZSS L1 C/A, GLONASS L10F
  - BeiDou B1
  - SBAS L1 C/A: WAAS, EGNOS, MSAS
- Navigation update rate:
  - Single GNSS up to 18 Hz
  - Concurrent GNSS up to 10 Hz
- Position accuracy: 2.0 m CEP
- Acquisition (MAX-M8Q,W/ MAX-M8C)
  - Cold starts: 26 s / 27 s
  - Aided starts: 2 s / 4 s
  - Reacquisition: 1.5 s / 1.5 s
- Sensitivity

  - Tracking & Nav: −167 dBm / −164 dBm
  - Cold starts: −148 dBm / −147 dBm
  - Hot starts: −156 dBm / −156 dBm
- Assistance
  - AssistNow GNSS Online
  - AssistNow GNSS Offline (up to 35 days, requires host integration)
  - AssistNow Autonomous (up to 6 days)
  - OMA SUPL & 3GPP compliant
- Oscillator
  - TCXO(MAX-M8Q/M8W),
  - Crystal (MAX-M8C)
- RTC crystal: built-In (MAX-M8Q/M8W) or cost efficient solution with higher Backup current (MAX-M8C)
- Anti jamming: active CW detection and removal
- Memory: Onboard ROM
- Supported antennas: Active and passive
- Support tools:
  - EVK-M8N: u-blox M8 GNSS evaluation kit
  - with TCXO, supports MAX-M8Q/M8W
  - EVK-M8C: u-blox M8 GNSS evaluation kit
  - with crystal, supports MAX-M8C

2.6 Data Logging

Data logging is done with a a simple microSD adapter breakout board. The board uses a MM74HC4050 hex logic level down converter. https://www.adafruit.com/products/254

- Onboard 5 V to 3 V regulator provides 150 mA for power-hungry cards
- 3 V level shifting means you can use this with ease on either 3 V or 5 V systems
- Uses a proper level shifting chip, not resistors: less problems, and faster read/write access
- Use 3 or 4 digital pins to read and write 2Gb+ of storage!
- Activity LED lights up when the SD card is being read or written

- Four #2 mounting holes
- Push-push socket with card slightly over the edge of the PCB so its easy to insert and remove

2.6.1 Technical Details

- Dimensions (assembled): 31.85 mm x 25.4 mm x 3.75 mm
- Weight: 3.43 g

2.6.2 Storage

A 4 GB micro-SD SDHC card is used as storage. For easy of utilisation it is recommended to format the micro SD card with the tool provided by ScanDisk, see https://www.sdcard.org/downloads/formatter_4/ and https://www.adafruit.com/products/102.

Every logged line is about 189 bytes large, this allows to store approximately $21 \cdot 10^6$ lines.

2.7 Temperature and Humidity

This I2C (I2C address of 0x40) digital humidity sensor HTU21D-F, has a typical accuracy of ±2 %RH with an operating range that's optimized from 5 %RH to ±95 %RH. Operation outside this range is still possible - just the accuracy might drop a bit. The temperature output has an accuracy of ±1 °C from −30 to 90 °C.

The breakout board includes the filtered version (the white bit of plastic which is a PTFE filter to keep the sensor clean), a 3.3 V regulator and I2C level shifting circuitry. This lets you use it safely with any kind of microcontroller with 3.3 V to 5 V power or logic. https://www.adafruit.com/products/1899

2.8 Pressure and Altitude

This pressure sensor from Freescale is a great low-cost sensing solution for precision measurement of barometric pressure and altitude. The MPL3115A2 has a typical 1.5 Pa resolution, which can resolve altitude at 0.3 m. It has some upsides compared to the BMP180, such as interrupt outputs for ultra-low power usage, and its also a heck of a lot easier to read altitude with a built in altimeter calculation - no calibration reading and calculating required. As a bonus, there's even a fairly good temperature sensor with ±1 °C typical accuracy (±3 °C maximum).

This chip likes to be used with 2 V to 3.6 V power and logic voltages, so we placed it on a breakout with a 3 V regulator and logic level shifting. https://www.adafruit.com/products/1893

2.8.1 Technical Details

- Dimensions: 18 mm x 19 mm x 2 mm
- Weight: 1.2 g
- Vin: 3 V to 5.5 V
- Logic: 3 V to 5 V compliant
- Pressure sensing range: 50×10^3 Pa to 110×10^3 Pa
- This board/chip uses I2C 7-bit address 0x60

2.9 IMU 10DOF

This inertial-measurement-unit combines 3 of the best quality sensors available on the market to give you 11 axes of data: 3 axes of accelerometer data, 3 axes gyroscopic, 3 axes magnetic (compass), barometric pressure/altitude and temperature.

The L3DG20H gyroscope, LSM303DLHC accelerometer compass and BMP180 barometric/temperature sensors are all on one breakout here. Since all of them use I2C, you can communicate with all of them using only two wires. Most will be pretty happy with just the plain I2C interfacing, but we also break out the 'data ready' and 'interrupt' pins, so advanced users can interface with if they choose. There's level shifting circuitry so the IMU can be used with 3 V or 5 V logic boards.

2.9.1 Technical Details

- L3GD20H 3-axis gyroscope: ±250, ±500, or ±2000 $^\circ\,s^{-1}$ scale
- LSM303 3-axis compass: ±1.3 to ±8.1 G magnetic field scale
- LSM303 3-axis accelerometer: ±2, ±4, ±8 or ±16 g selectable scale
- BMP180 barometric pressure/temperature: −40 to 85 °C, 30×10^3 to 110×10^3 Pa range, 0.17 m resolution

Dimensions (without header):

- Length: 38 mm

- Width: 23 mm
- Height: 3 mm
- Weight: 2.8 g
- This board/chip uses I2C 7-bit addresses 0x19 & 0x1E & 0x6B & 0x77

2.9.2 Roll, Picth and Heading Calculation

Roll γ, pitch ρ and heading ψ are calculated with following formulas, based on AN3192 application note *Using LSM303DLH for a tilt compensated electronic compass* found on www.st.com. The LSM303DLH accelerometer raw measurements are a_x, a_y and a_z, after applying the according calibration the values a_x, a_y have to be normalized in order to calculate roll γ and pitch ρ. The normalized values a_{x1} and a_{y1} are;

$$a_{x1} = \frac{a_x}{\sqrt{a_x^2 + a_y^2 + a_z^2}} \tag{2}$$

and

$$a_{y1} = \frac{a_y}{\sqrt{a_x^2 + a_y^2 + a_z^2}}. \tag{3}$$

$$pitch = \rho = \arcsin\left(-a_{x1}\right) \tag{4}$$

$$roll = \gamma = \arcsin\left(\frac{a_{y1}}{\cos\rho}\right) \tag{5}$$

Roll γ value is expressed in $-90°$ to $90°$ and pitch ρ values are expressed in $-90°$ to $90°$.

The LSM303DLH magnetic sensor raw measurements are m_x, m_y and m_z. As the earth magnetic field may be distorted by hard and soft iron distortions, the values have to be calibrated. Further details can be found at https://github.com/ptrbrtz/razor-9dof-ahrs/wiki/Tutorial.

$$\begin{bmatrix} m_{x0} \\ m_{y0} \\ m_{z0} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \times \begin{bmatrix} m_x - m_{x,off} \\ m_y - m_{y,off} \\ m_z - m_{z,off} \end{bmatrix} \tag{6}$$

Where values $a_{i,j}$ $i, j = 1, 2, 3$ and $m_{k,off}$ $k = x, y, z$ are found by analysing the distorted measurements. Details and the underlying Matlab function can be found under the above stated URL.

Again the value have to be normalised in order to calculate the heading ψ. The normalised values m_{x1}, m_{y1} and m_{z1} are;

$$m_{x1} = \frac{m_{x0}}{\sqrt{m_{x0}^2 + m_{y0}^2 + m_{z0}^2}}, \tag{7}$$

$$m_{y1} = \frac{m_{y0}}{\sqrt{m_{x0}^2 + m_{y0}^2 + m_{z0}^2}} \tag{8}$$

and

$$m_{z1} = \frac{m_{z0}}{\sqrt{m_{x0}^2 + m_{y0}^2 + m_{z0}^2}}. \tag{9}$$

The tilt compensated magnetic sensor measurements m_{x2}, m_{y2} and m_{z2} can be obtained as:

$$m_{x2} = m_{x1} \cos \rho + m_{z1} \sin \rho, \tag{10}$$

$$m_{y2} = m_{z1} \sin \gamma \sin \rho + m_{y1} \sin + m_{z1} \cos \gamma \cos \rho \tag{11}$$

$$heading = \psi = \arctan \left(\frac{m_{y2}}{m_{x2}} \right). \tag{12}$$

Heading ψ value is expressed in 0 °North to 359°. Definition of the coordinate system can be found din Fig. 8.

NB. text directly taken from AN3192 application note:

Because the accelerometer measurements cannot distinguish earths gravity from linear acceleration or angular acceleration, fast motion causes pitch/roll calculation error which then directly introduces an error to the heading calculation. In most cases, the fast motion doesn't last long and the device goes back to a stationary position. So the heading accuracy in an electronic compass means static accuracy when the device is still or without acceleration.

The above formula are implemented in the code starting at Line 572 in List. 3.

2.9.3 Calibration of Gyroscope

The gyroscope is calibrated against any offset by taking the average of the gyroscopic readings Ω_x, Ω_y and Ω_z over a sufficient long period of without moving the sensor. The average is then used to correct the gyroscope readings. https://www.adafruit.com/products/1604

2.10 Luminosity and Light Sensor

The TSL2561 luminosity sensor is an advanced digital light sensor, ideal for use in a wide range of light situations. Compared to low cost CdS cells, this sensor is more precise, allowing for exact lux calculations and can be configured for different gain/timing ranges to detect light ranges from up to 0.1 to 40×10^3 lx. The sensor contains both infrared and full spectrum diodes, that means it can separately measure infrared, full-spectrum or human-visible light.

The sensor has a I2C interface. The built in ADC means you can use this with any micro controller, even if it doesn't have analog inputs. The current draw is extremely low, so its great for low power data-logging systems, about 0.5 mA when actively sensing, and less than 15 µA when in power down mode. https://www.adafruit.com/products/439

2.10.1 Technical Details

- Approximates Human eye Response
- Precisely Measures Illuminance in Diverse Lighting Conditions
- Temperature range: −30 to 80 °C
- Dynamic range: 0.1 to 40×10^3 lx
- Voltage range: 3 V to 5 V
- This board/chip uses I2C 7-bit addresses 0x39 (or 0x29, 0x49, selectable with jumpers)

2.11 High Accuracy Temperature Sensor

This I2C digital temperature sensor MCP9808 is one of the more accurate/precise we've ever seen, with a typical accuracy of ±0.25 °C over the sensor's −40 to 125 °C range and precision of 0.0625 °C. They work great with any micro controller using standard i2c. There are 3 address pins so you can connect up to 8 to a single I2C bus without address collisions. https://www.adafruit.com/products/1782

2.11.1 Technical Details

- ±0.25 °C typical precision over −40 to 125 °C range (±0.5 °C guaranteed max from −20 to 100 °C) 0.0625 °C resolution.
- Operating Current: 200 µA (typical)
- Sensing Temperature: −40 to 125 °C
- Accuracy: ±0.25 °C typical
- Voltage - Supply: 2.7 V to 5.5 V
- Operating Temperature: −40 to 125 °C
- Uses any I2C address from 0x18 thru 0x1F
- Dimensions: 21 mm x 13 mm x 2 mm
- Weight: 0.9 g

2.12 Power Supply

The power is supply through the TPS61201 on the HABduino shield. The power consumption is estimated based on a long term test measurement. Fig. 5 shows the logged power consumption for the fully assembled and operating flight computer.

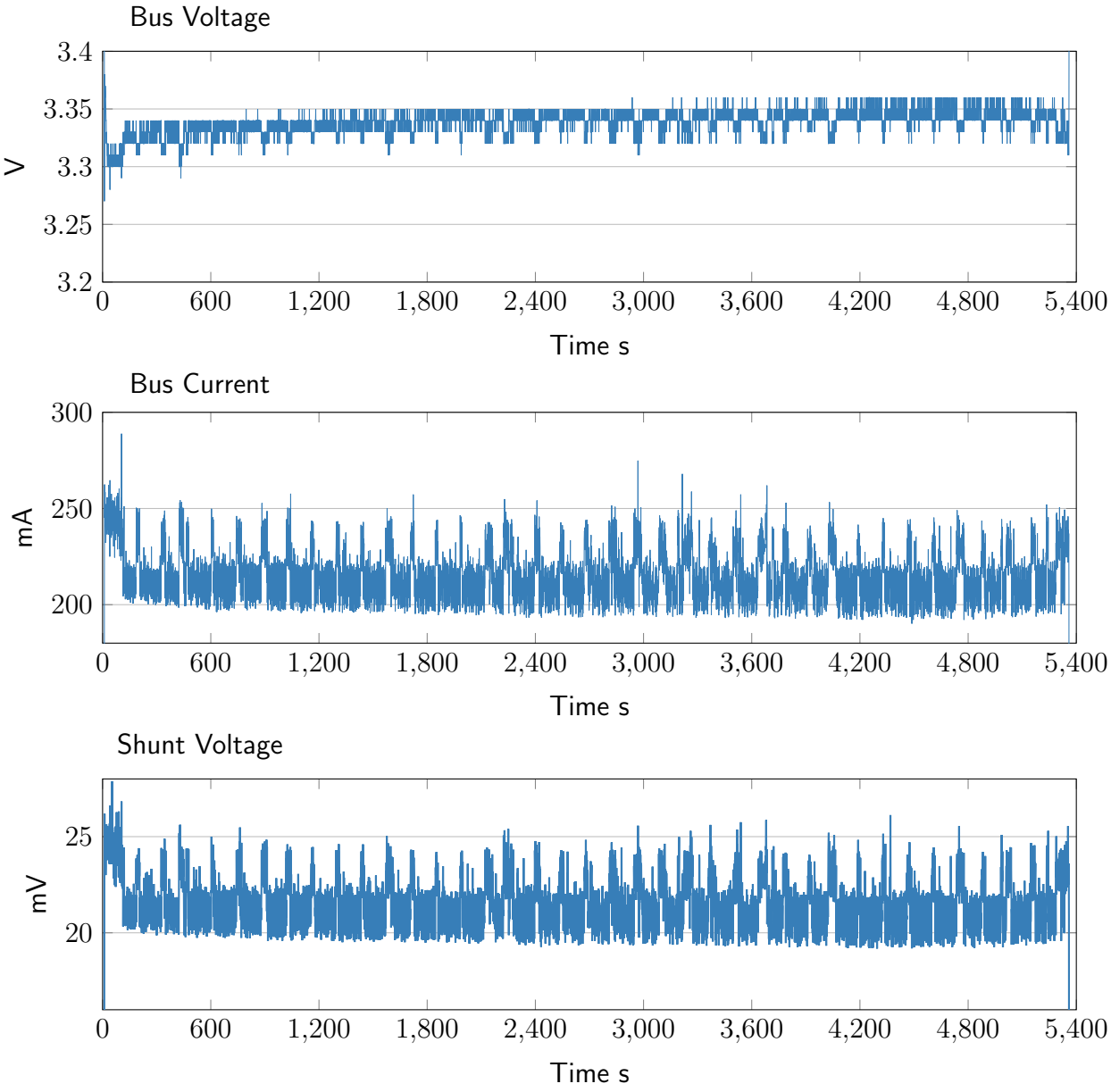


Figure 5: Logged power consumption for the assembled and fully operating flight computer.

The power consumption was measured with an INA29 current power monitor. The measurement was done at ambient temperature of around 20 °C using a fairly new Saft LSH 20 battery as power source.

The power consumption is around 220 mA, leading to a LSH 20 battery life of around

40 h at 20 °C. However, the capacity of the battery drops to 5.8 A h at a cell temperature of −40 °C. This would lead to a battery life of around 23 h. All calculations are done based on a conservative power consumption of 250 mA. Further, the cell voltage remains above 2.8 V even for temperatures down to −40 °C. This should allow the low input voltage synchronous boost converter TPS61201 to operate efficiently. However, it is clear that the LSH 20 is over sized as power source, however it guaranties optimal and secure operation even in very cold environment.

3 Flight Computer Software

The flight computer software was programmed using the Arduino IDE 1.6.5 As basis of the flight computer software the code made available on http://habduino.org/ was used. This code is basically handling all transmission activity. However, all part regarding the additional sensors and logging to the micro-SD card was added. Further, the original code was developed to function with the Arduino UNO, thus the PWM timers had to be changed in order to function with the Arduino Mega 2560 R3. Specification can be found at http://playground.arduino.cc/Main/TimerPWMCheatsheet.

3.1 Stability and Reliability

In order to guaranty stability and reliability, the Watchdog function of the Atmel ATmega microcontroller is used. The Atmel ATmega microcontroller has a programmable Watchdog Timer with Separate On-chip Oscillator. The basic principle of the Watchdog is to restart the micrcontroller, if the Watchdog timer is not reset within specific time interval (in our case 8 s), i.e. the microcontroller hangs. The time reset is done with the command `wdt_reset()` in the flight computer software.

3.2 Modifications

The MPL3115A2 pressure sensor showed some initial problems when pressure and altimeter data were read. This glitch was solved by correcting the *MPL3115A2.cpp* file. However, the altitude is not taken directly from the chip, but rather calculated with the formula stated in the data sheet, see also Tab. 1, with the offset set to 0. In a later version this should not be the case any more.

Initially some problems were observed regarding the I2C communication. During test phase individual wires from the sensors were disconnected. This to simulate possible cable or soldier as well as chip failures. During these test a problem in the two wire interface (TWI) was found. Basically it could lead to the microcontroller to end up in a infinite loop as the TWI was waiting for an answer from the device. A description and solution of this behaviour can be found here, http://forum.arduino.cc/index.php?topic=19624.0. Initially the fixes were made to *twi.h* and *twi.c*, but with the new version of Arduino IDE

1.6.1 not repeated again. Now it is only the watchdog to restart the microcontroller once hanging.

3.3 Known Issues

The data are written to the microSD during the `ISR(TIMER5_COMPA_vect)` timer call. This is however not optimal, as using the SPI is not recommended during a timer call. In this particular case, however no consequences can be observed as the `writeToSD()` function is call during two transmissions. The initial idea is to have the possibility to pause in-between two transmissions, see Line 689 in List. 3.

4 Image Capturing

The image capturing is done with a GoPro HERO4 Silver featuring 1080p60 and 720p120 video, 12MP photos up to 30 frames per second. In order to extend the power supply the Battery BacPac was added to the GoPro camcorder. http://gopro.com/

In contradiction with some statements found in the world wide web, the GoPro is mounted inside the water tide housing. In order to remove all moisture inside the housing the housing is cleaned with Dust-Off (1,1-Difluoroethane) and then filled with 1,1-Difluoroethane while the camera lens is pointing down (1,1-Difluoroethane is heavier than air). No anti moist inserts are added.

During the flight a 12MP (4000x3000 pixels 72 DPI) photo is captured every seconds as the resolution of images are higher than those of videos captured with the GoPro camcorder. Thus, 60 photos a minute or 3600 photos an hour.

A microSD card with class 10 rating of 64 GB size is used. With an average image size of 2.3 MB a total of around 27 800 images can be saved on the mircoSD card. This would correspond to capture images for up to 7.5 h.

4.1 Technical Details

Only technical details of relevance are presented below.

Weight Camera 83 g

Camera with housing 147 g

Video Resolution Frames Per Second (fps) NTSC/PAL 15, Field of View (FOV) Ultra Wide, Screen Resolution 3840x2160.

Video Format H.264 codec, .mp4 file format

Video and Photo Record video and capture Time Lapse photos at the same time. Available intervals are 5, 10, 30 and 60 seconds. Video Resolution 1440p, Video Frames per Second (fps) 24, Video FOV Ultra Wide.

SuperView Mode 1080p SuperView, 1920 x 1080

Photo Resolutions Resolution 12MP, Field of View (FOV) Wide and Screen Resolution 4000 x 3000.

Time Lapse Photo Intervals 0.5, 1, 2, 5, 10, 30 and 60 s

Battery and Charging Rechargeable lithium-ion battery, rated at 1160 mA h, 3.8 V, 4.4 W h

Battery Life Approximate continuous recording time (hr:min) you can expect when shooting in various video modes using a fully charged battery. With Wi-Fi Off + Using Battery BacPac. Video Mode 1080p 30fps and SuperView estimated with 3:05.

Audio Format: 48kHz sampling rate, AAC compression. Advanced AGC (automatic gain control) with multi-band compressor

Storage microSD memory card with a Class 10 or UHS-1 rating required. See list of recommended microSD cards. Up to 64 GB capacity supported. Record times vary with resolutions and frame rates.

Connect via mini USB to USB cable (included), or copy files from the microSD card to your computer.

5 Tracking

In this section the tracking approach is outlined and its individual components explained. The general idea is to use a Software Defined Radio (SDR) and decode the received signal with the help of a personal computer. The computer further, assists while tracking with position visualization using online map services.

5.1 Software Defined Radio

As Software Defined Radio (SDR) a FUNcube Dongle Pro+ is used. Its sensitivity should allow for precise tracking of the HAB during flight. More details on operation are found on http://www.funcubedongle.com. Some general technical specifications are listed below.

- Guaranteed frequency range: 150 kHz to 240 MHz and 420 MHz to 1.9 GHz
- Typical frequency coverage: 150 kHz to 260 MHz and 410 MHz to 2.05 GHz

- TCXO specified at 0.5 ppm (in practice about 1.5 ppm)
- 192 kHz sampling rate
- Extremely simple hardware setup, just two connections, an in and an out:
  - Standard SMA female antenna port
  - USB 1.x type A male connection
- Eleven discrete hardware front end filters including:
  - 6 MHz 3 dB bandwidth (10 MHz at −40 dB) SAW filter for the 2 m band.
  - 20 MHz 3 dB bandwidth (42 MHz at −40 dB) SAW filter for the 70 cm band
  - Third- and fifth-order LC bandpass filters for other bands
- Front end LNA OIP3 30 dB
- Typical noise figures:
  - 50 MHz 2.5 dB
  - 145 MHz 3.5 dB
  - 435 MHz 3.5 dB
  - 1296 MHz 5.5 dB
- Typical NFM 12dB SINAD measurements:
  - 145 MHz 0.15 µV
  - 435 MHz 0.15 µV
- No additional drivers required for Linux, OSX or Windows
- Integrated 5 V bias T switchable from software

5.2 Antenna

As recipient antenna a **YAGI-434A** from LPRS, http://www.lprs.co.uk/wireless-modules/antennas/yagi-antennas.html is used. Its electrical specifications are

Frequency Range 390-480 MHz

Nominal Impedance 50 Ω

Gain 9 dBi at 450 MHz

VSWR 1.5:1

F/B Ratio >13 dB

Maximum Input Power 100 W

Polarization Vertical

Connector SMA male

Whereas the mechanical specifications are

Support Boom Material Steel Bracket

Element Material Aluminium

Number of elements 7

Antenna dimensions 1000x400x70 mm

Antenna weight 575 g

Ambient temperature −40 °C to 60 °C

The FUNcube Dongle Pro+ is directly mounted on the antenna and connected via a 2 m USB cable to the computer.

5.3 Software

In order to track the HAB flight two programs have to be run simultaneously on a computer. Both programs a shortly described below.

5.3.1 Fldigi

The first an most important program is Fldigi 3.22.12, the documentation can be found here http://www.w1hkj.com/FldigiHelp-3.22/index.html. Taken from the webpage:

Fldigi is a computer program intended for Amateur Radio Digital Modes operation using a PC (Personal Computer). Fldigi operates (as does most similar software) in conjunction with a conventional HF SSB radio transceiver, and uses the PC sound card as the main means of input from the radio, and output to the radio. These are audio-frequency signals. The software also controls the radio by means of another connection, typically a serial port. Fldigi is multi-mode, which means that it is able to operate many popular digital modes without switching programs, so you only have one program to learn. Fldigi includes all the popular modes, such as DominoEX, MFSK16, PSK31, and RTTY.

Additionally DSP Radio 1.4.2 is used to set the frequency and check the signal from the FUNcube Dongle Pro+. For further details see http://dl2sdr.homepage.t-online.de.

In short, Fldigi decodes the received signal based on the setting and logs the received characters into a local file.

5.3.2 Flight Control Centre

The Flight Control Centre is a web based application written in html making use of javascript. The complete source code can be found in appendix B. The application request a internet connection for map visualisation and the possibility to locate its location.

The Flight Control Centre reads the log file generated by Fldigi (see section above for further explanation) every half second and displays the current received string. If a complete string is received, it is analysed and the data visualised. Further, the current transmitted position together with the current position of the computer running the Flight Control Centre are displayed on a map. This should allow to track the HAB with easy.

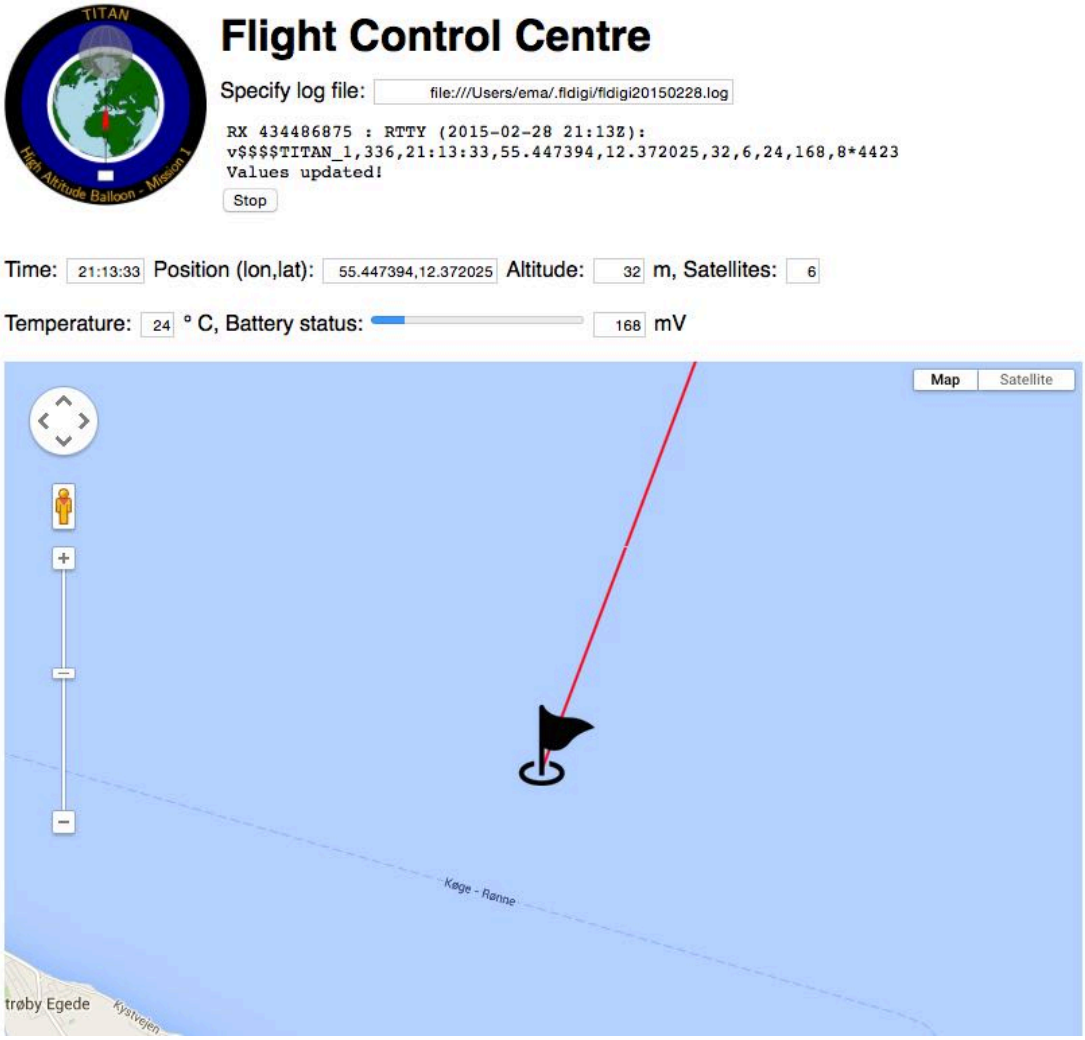


Figure 6: Visualisation of the upper part of the Flight Control Centre.

Fig. 6 shows the upper part of the Flight Control Centre. After a field specifying the log file path and name, the currently read string is shown. The button below starts or stops the automatic updating. The following two lines show the values form the last completed received string. On the map the current location of the HAB is shown, flag symbol. A

straight red line connects the HAB position with the current location of the computer running the Flight Control Centre.

On the lower part the current state of the GPS module and the individual sensors as well as the logging status is shown. The current states are obtained from the transmitted *errorstatus*.

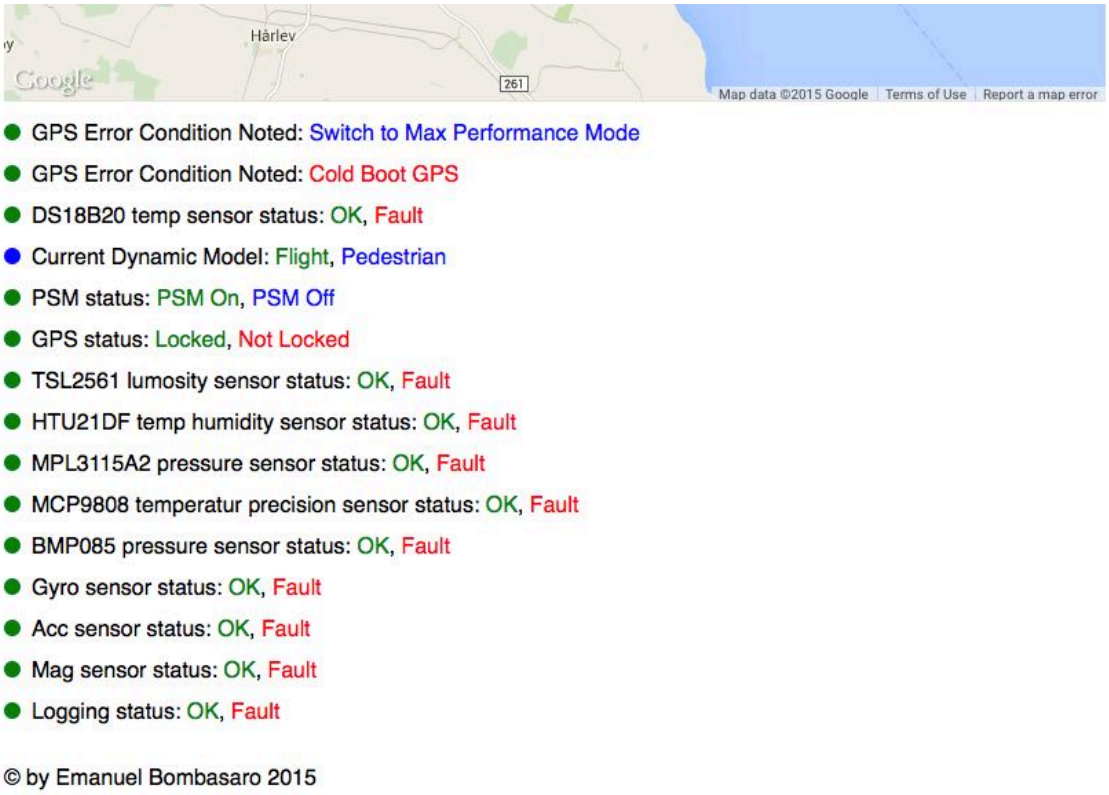


Figure 7: Visualisation of the lower part of the Flight Control Centre. Note the lower part of the map at the top.

The Flight Control Centre was develop to work with Safari 8 https://www.apple.com/safari/ or higher, but should work with any other web browser on any other system as well.

6 Balloon, Parachute and Payload Box

The core part of the design is the HAB hardware, consisting of the payload box, the balloon and parachute. This section describes the individual parts and design fundamentals.

6.1 Payload Box

The payload box is constructed of a foamed polystyrene box, with an external length of 225 mm, a width of 225 mm and a height of 195 mm. The internal dimensions are 165 mm, a width of 165 mm and a height of 130 mm respectively. The foamed polystyrene guarantees good thermal isolation and shock absorption during landing impact. The individual sensors are mounted by generating small pits in which they are fixed by tape. For the camera objective a hole is cut, such that the camera is looking with approximately 30° from the horizontal downward. Fig. 8 shows schematically the location of the individual sensors, antenna and camera. The flight computer and battery is located inside the box and fixed with goffered plastic packing film.

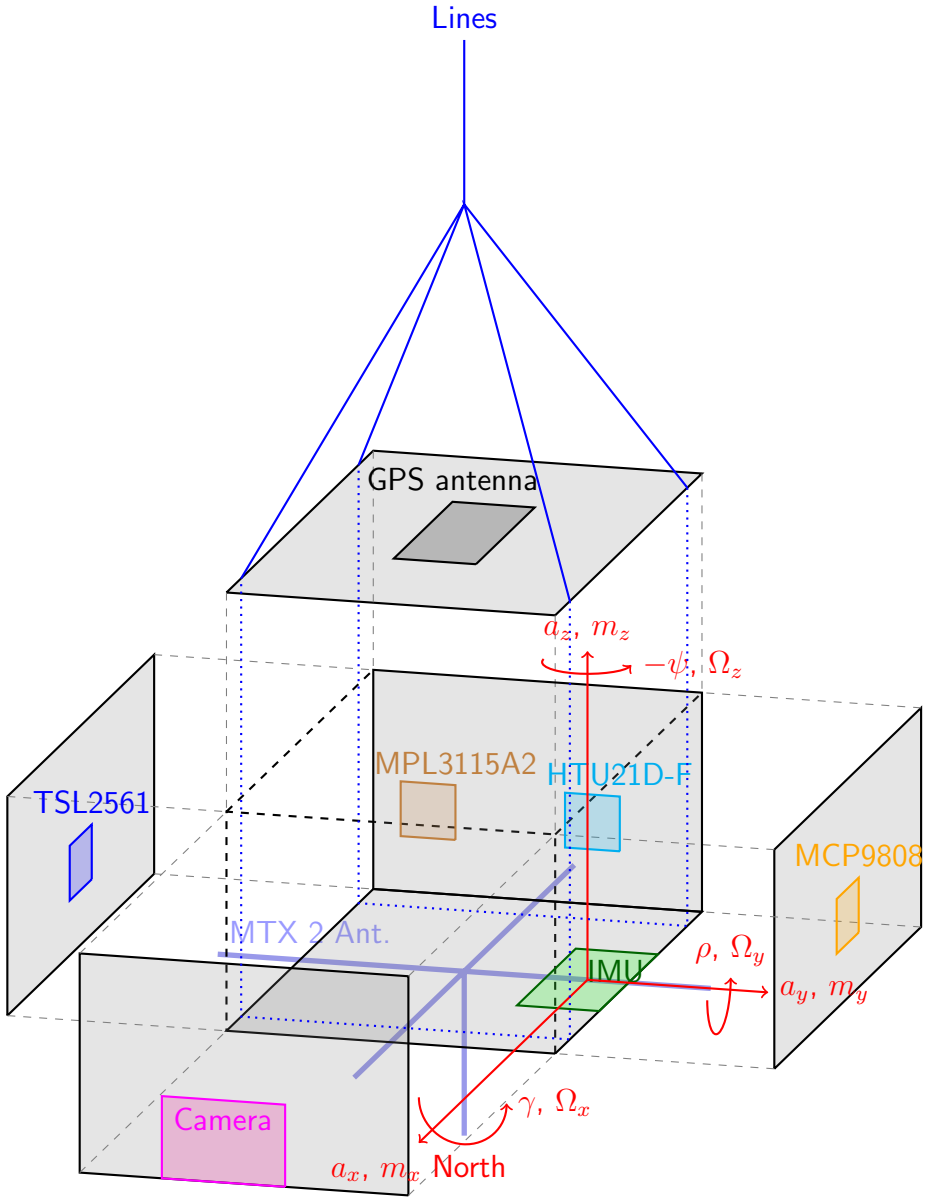


Figure 8: Schematic of payload box with sensor location and coordinate system of the IMU sensor. The transmission antenna and line routing are further shown.

Fig. 9, Fig. 10 and Fig. 11, show the partly assembled payload box.

Figure 9: Payload box seen from top with open hatch, flight computer already wrapped in goffered plastic packing film.

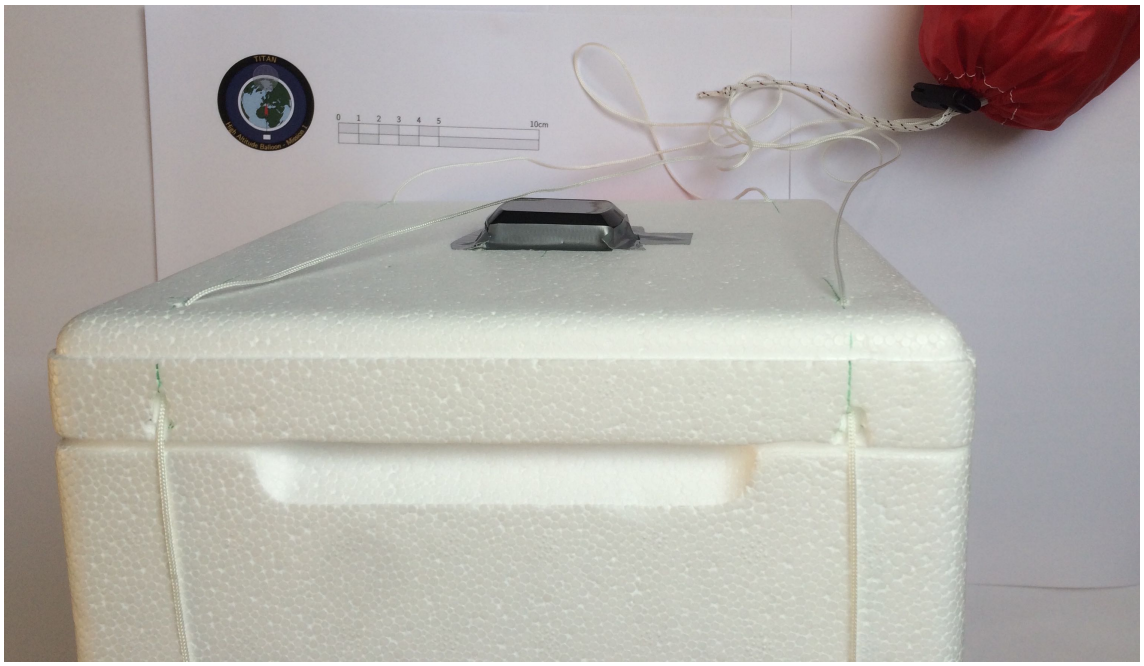


Figure 10: Closed payload box, the antenna on top is visible and the routing of the lines through the hatch.

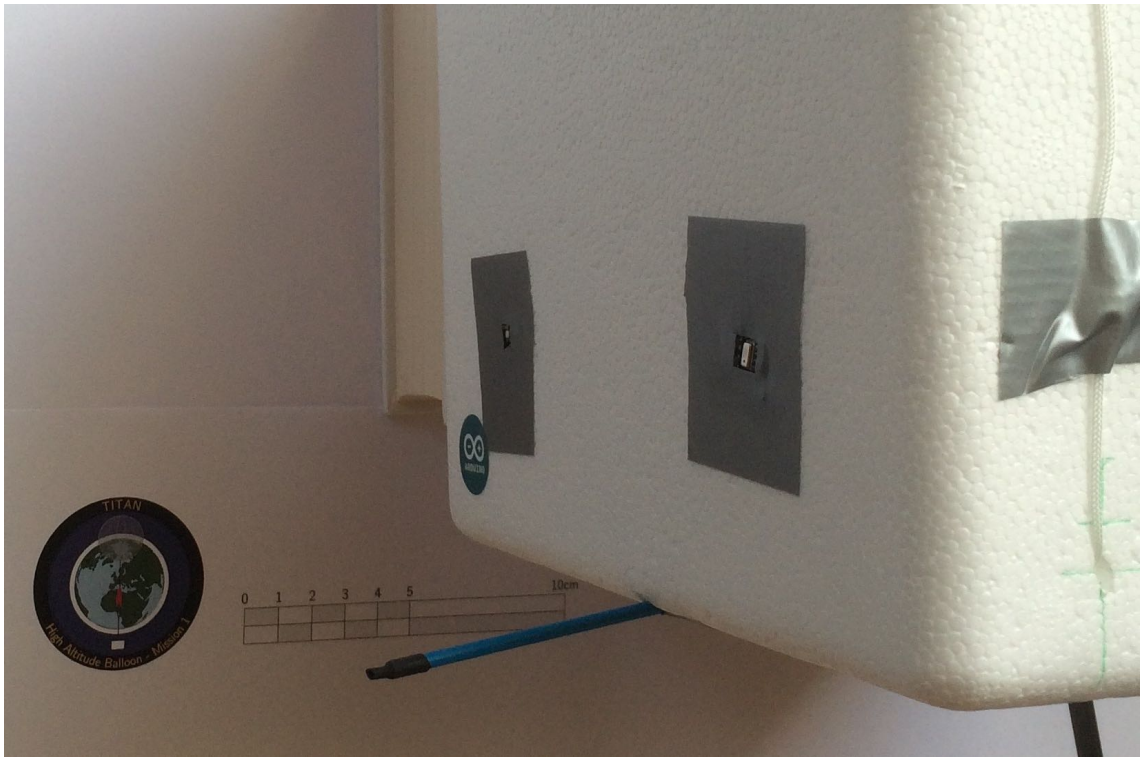


Figure 11: Closed payload box, side view with the HTU21D-F (below) and MPL31115A2 (top) sensors. The line routing can be seen on the right border.

6.1.1 Payload label

The payload label is intended to identify the HAB as such if found by a third party not aware of its content. Further, it seals the payload box to avoid accidental opening during flight and landing. One payload box was put inside the box and two where used to seal the hatch to the box on the outside.

Figure 12: Payload label

6.1.2 Payload Box Weight

The payload box weight is crucial for flight performance (peak altitude and ascent descent velocity) and thus has to be identified with care. The process involved configuration considerations and continuous modification till reaching an optimum weight. Tab. 4 lists the weight of the individual components.

Part	Weight
Flight computer without transmission antennas, but with GPS antenna.	155 g
Battery with cable.	102 g
GoPro including all parts, i.e. housing, power back.	191 g
Empty payload box, but including holes for individual components and transmission antenna.	190 g
Lines and parachute.	62 g
Sum part not assembled	700 g
Fully assembled payload box ready to lift off including lines, swivel and parachute. Only missing part is the balloon.	717 g

Table 4: Measured weight of individual components and total payload weight. All values apart from the sum are measured values.

The area weight of the payload box (smallest area is 22.5x19.5 cm) is $1.64\,\mathrm{g\,cm^{-2}}$ for a payload weight of 720 g. If the balloon is included the total weight increase to 2220 g and thus leads to a area weight of $5.06\,\mathrm{g\,cm^{-2}}$.

6.2 Variables for Calculation

Tab. 5 summaries the variable used for the calculation for choosing the balloon and parachute. As can be noted for air and helium the densities are taken at different temperature levels and with a pressure similar to atmospheric pressure at surface. This in order to obtain a most realistic value for the balloon at launch by still keeping some conservatism.

Value	Variable	Value
Density Helium at 0 °C and 101 kPa	ρ_{He}	$0.1786\,\mathrm{kg\,m^{-3}}$
Density Air at 20 °C and 101 kPa	ρ_{Air}	$1.2041\,\mathrm{kg\,m^{-3}}$
Density difference $\rho_{Air} - \rho_{He}$	$\dot{\rho}$	$1.0255\,\mathrm{kg\,m^{-3}}$

Gravitational acceleration	g	$9.806\,65\,\mathrm{m\,s^{-2}}$
Ideal (universal) gas constant	R	$8.314\,47\,\mathrm{J\,mol^{-1}\,K^{-1}}$
Molar mass of Earth's air	M	$0.028\,964\,4\,\mathrm{kg\,mol^{-1}}$
Sea level standard temperature	T_0	$288.15\,\mathrm{K}$
Temperature lapse rate	L	$0.0065\,\mathrm{K\,m^{-1}}$
Payload mass including parachute	m_P	$0.720\,\mathrm{kg}$
Ballon mass, HAB-1500	m_B	$1.500\,\mathrm{kg}$
Ballon air drag, HAB-1500	C_d	$0.25-$
Brust diameter, HAB-1500	d_B	$9.45\,\mathrm{m}$
Parachute diameter, 36" Spherachute LT	d_P	$0.914\,\mathrm{m}$
Parachute air drag, 36" Spherachute LT	$C_{d,P}$	$0.78-$

Table 5: Variables for calculation for balloon and parachute.

6.3 Balloon

The burst altitude H_B and ascend rate v_a can be estimated using simple formula. In Tab. 5 the variables needed for the calculation are listed.

6.3.1 Burst Altitude

The burst altitude can be estimated start using the atmospheric density equation,

$$\frac{\rho}{\rho_0} = \exp\left[-\frac{gMH_B}{RT_B}\right], \tag{13}$$

resulting in

$$H_B = -\frac{RT_B}{gM}\ln\left[\frac{\rho}{\rho_0}\right]. \tag{14}$$

Assuming the temperature of the helium and the air outside the balloon remains in equilibrium we can write

$$H_B = -\frac{RT_B}{gM}\ln\left[\frac{d_L^3}{d_B^3}\right], \tag{15}$$

with d_L the launch diameter of the balloon as the density is a linear function of the volume V and further a cubic of the diameter d. The balloon is assumed to be a perfect sphere and thus the volume is

$$V_i = \frac{4}{3}\,\pi\left(\frac{d_i}{2}\right)^3 \tag{16}$$

The temperature T_B at burst altitude can be approximated with $247.28\,\mathrm{K}$. Based on the International Standard Atmosphere model this corresponds to an altitude of $38.7\times10^3\,\mathrm{m}$.

6.3.2 Ascend Rate

The balloon gross lift G_L at launch in kg is

$$G_L = \dot{\rho}\ \frac{4}{3}\ \pi \left(\frac{d_L}{2}\right)^3 \tag{17}$$

and thus the free lift F_L in N

$$F_L = (G_L - m_P - m_B)\, g. \tag{18}$$

The ascent rate v_a in $\mathrm{m\,s^{-1}}$ can be estimated with

$$v_a = \sqrt{\frac{2\ F_L}{C_d\ \rho_{Air}\ \pi\,(d_L/2)^2}}, \tag{19}$$

assuming a constant rate as the balloon cross section increases with decreasing atmospheric pressure. The ascent time t_a in s is approximately

$$t_a = \frac{H_B}{v_a}. \tag{20}$$

If the ascent rate is smaller then $4.8\,\mathrm{m\,s^{-1}}$ the balloon can result in a floater.

6.3.3 Selection

Based on the formula outlined in the previous section and the values in Tab. 5 the burst altitude H_B and ascend rate v_a can be estimated. One of the major parameters remains the diameter at launch d_L defining the launch volume V of the balloon. Fig. 13 shows the functional dependency between the burst altitude H_B, ascend rate v_a, ascend time t_a and the launch volume V_L.

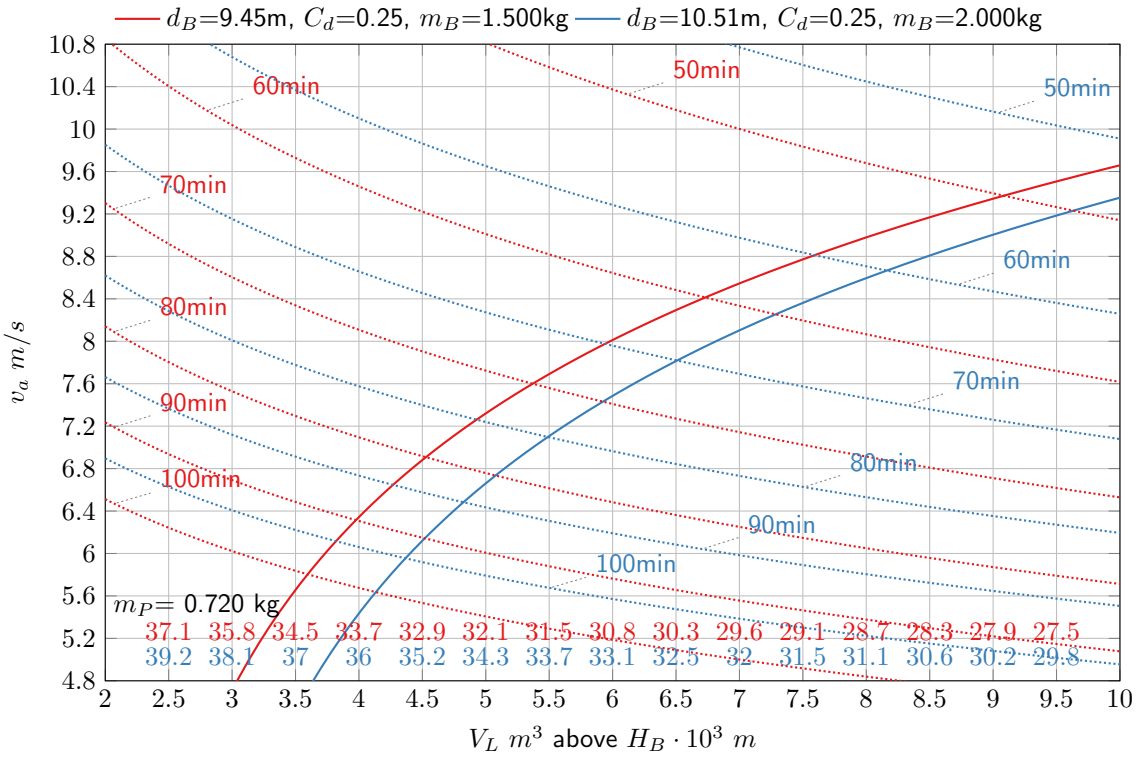


Figure 13: Functional dependency of launch volume V_L and or burst altitude H_B with ascend rate v_a. The gray lines are isochrones for ascend time t_a.

From Fig. 13 it seams reasonable to select a **HAB-1500** balloon from http://kaymontballoons.com/Near_Space_Photography.html. The values of the balloon can be found in Tab. 5.

The burst altitude results in

$$H_B = -\frac{RT_B}{gM} \ln\left[\frac{d_L^3}{d_B^3}\right] = -\frac{RT_B}{gM} \ln\left[\left(\frac{1.877m}{9.45m}\right)^3\right] = 35104m, \tag{21}$$

with a launch volume of

$$V_L = \frac{4}{3}\,\pi \left(\frac{d_L}{2}\right)^3 = \frac{4}{3}\,\pi \left(\frac{1.877m}{2}\right)^3 \approx 3.460m^3. \tag{22}$$

For the calculation the volume was set and then the launch diameter d_L calculated, this is why the equation above is not exact. The calculations are made with higher precision and thus not affected. The temperature at 35 104 m is approximately 237.34 K based on the International Standard Atmosphere model. Thus, the assumed 247.28 K for T_B are acceptable exact. However it may lead the balloon not to reach the estimated burst altitude.

The balloon gross lift G_L at launch is

$$G_L = \dot{\rho}\,\frac{4}{3}\,\pi \left(\frac{d_L}{2}\right)^3 = 1.0255kgm^{-3} \cdot 3.460m^3 = 3.548kg \tag{23}$$

and thus the free lift F_L

$$F_L = (G_L - m_P - m_B)\, g = (3.548kg - 0.720kg - 1.500kg)\, g = 13.03N. \tag{24}$$

The balloon is stable with an attached load of 2048 g, this to check the right launch volume V_L.

The ascent rate v_a results to

$$v_a = \sqrt{\frac{2\ F_L}{C_d\ \rho_{Air}\ \pi\,(d_L/2)^2}} = \sqrt{\frac{2 \cdot 13.03N}{0.25 \cdot 1.2041 \cdot \pi\,(1.877m/2)^2}} = 5.59ms^{-1}, \tag{25}$$

assuming a constant rate as the balloon cross section increases with decreasing atmospheric pressure. The ascent time t_a is approximately

$$t_a = \frac{H_B}{v_a} = \frac{35104m}{5.59ms^{-1}} = 6276s. \tag{26}$$

Thus, with a **HAB-1500** balloon a burst altitude H_b of 35 104 m should be reached. The initial fill volume V_L is 3.460 m^3. The ascend rate v_a is 5.59 m s^{-1} leading to an ascent time t_a of 105 min.

6.3.4 Helium and Filling Equipment

A 20 L tank is used containing 3.6 m^3 helium as pure product >99 %. The lift capacity is stated with 1 g per 1 L helium, which is in good agreement with the calculated balloon gross lift G_L see Eq. (23). The fill pressure of the tank is 200 bar. The filling tube with the pressure reducer and hose connected is shown in Fig. 14.

Figure 14: Filling tube with an outer diameter of 30 mm with hose and extra weight connected.

The filling tube has an outer diameter of 30 mm which matches the **HAB-1500** balloon neck diameter. In the filling tube knee an additional weight will be attached in order to test the balloon filling volume through the lift capacity. The weight of the filling tube with hose and extra weight has to be 2048 g.

The weight of filling tube including some part of the filling hose is around 230 g thus an additional weight of 1814 g is added to reach the requested 2048 g. The additional weight is made from a 2000 g scuba dive weight by drilling holes to adjust the weight to 1814 g.

6.4 Parachute

The descent velocity v_d of a parachute can be estimated with simple formula. Based on the descent velocity v_d the descent time t_d and a corresponding free fall height calculated.

6.4.1 Descent Velocity and Free Drop Height

The descent velocity v_d can be estimated with

$$v_d = \sqrt{\frac{2\ m_P\ g}{C_{d,P}\ \rho_{Air}\ \pi\ (d_P/2)^2}}, \tag{27}$$

where $C_{d,P}$ is the drag coefficient of the parachute and d_P is the diameter of the parachute in open configuration.

The decent time t_d result in

$$t_d = \frac{H_B}{v_d}. \tag{28}$$

For safety reason the kinetic energy of the payload box during descending can be estimated with The decent time t_d result in

$$E_P = \frac{v_d^2}{2} m_P. \tag{29}$$

When the payload box is dropped from the hight h_P without parachute the kinetic energy result in

$$E_{P,free} = \frac{\left(\sqrt{2gh_P}\right)^2}{2} m_P = g h_P m_P. \tag{30}$$

The corresponding free fall hight h_P can be found by setting $E_P = E_{P,free}$

$$h_P = \frac{v_d^2}{2g}. \tag{31}$$

6.4.2 Selection

A **36" Spherachute LT** parachute from https://spherachutes.storesecured.com/construction.asp is used to bring the payload box safely back to the ground. The parameters of

the parachute are listed in Tab. 5.

The descent velocity v_d results in

$$v_d = \sqrt{\frac{2\ m_P\ g}{C_{d,P}\ \rho_{Air}\ \pi\left(d_P/2\right)^2}} = \sqrt{\frac{2 \cdot 0.720 \cdot g}{0.78 \cdot 1.2041 \cdot \pi\left(0.914m/2\right)^2}} = 4.78ms^{-1} \tag{32}$$

and thus the descent time t_d in

$$t_d = \frac{H_B}{v_d} = \frac{35104m}{4.78ms^{-1}} = 7336s. \tag{33}$$

The corresponding free fall height is

$$h_P = \frac{v_d^2}{2g} = \frac{4.78^2}{2g} = 1.17m. \tag{34}$$

The **36" Spherachute LT** leads to a descend velocity v_d of $4.78\,\mathrm{m\,s^{-1}}$ and thus as descend time of approximately 122 min. The corresponding free fall height h_P is 1.17 m.

Fig. 15 shows the 36" Spherachute LT, with swivel (at the end of parachute) and lines mounted.

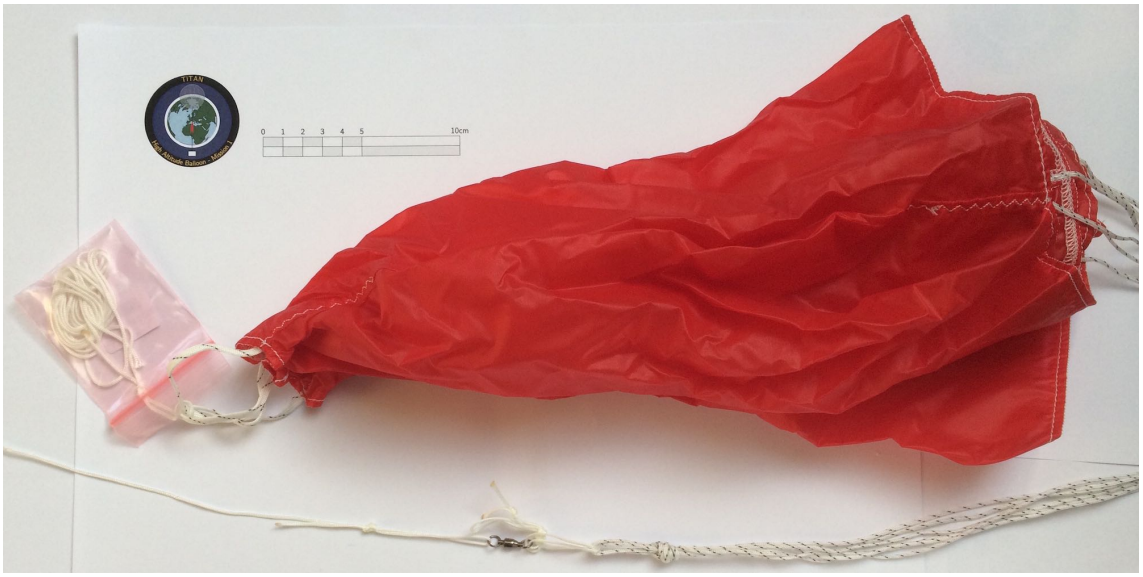
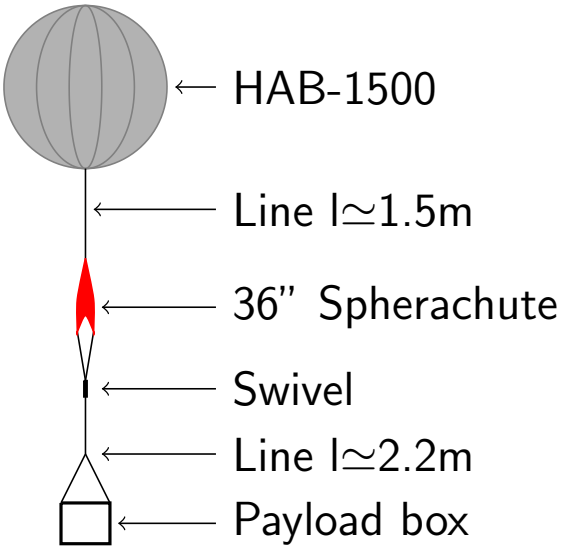


Figure 15: 36" Spherachute LT, with swivel and lines mounted.

6.5 Assembling and Technical Characteristics



Peak Altitude: 35 104 m

Ascent Velocity: 5.59 m s^{-1}

Descent Velocity: 4.78 m s^{-1}

Ascent Time: 105 min

Descent Time: 122 min

Total Flight Time: 227 min

Payload Weight: 0.720 kg

Length over all: ≃7 m

Balloon Volume at Launch: 3.460 m^3

Balloon Lift at Launch: 3.548 kg

Area Weight: 5.06 g cm^{-2}

The individual line lengths were chosen in order to comfort ascent and decent flight phase. The line between the payload box and the parachute is such to allow the parachute to deploy freely. The line between the parachute and the balloon is such that any possible remains of the burst balloon will hang below the parachute, but without interfering with the line below.

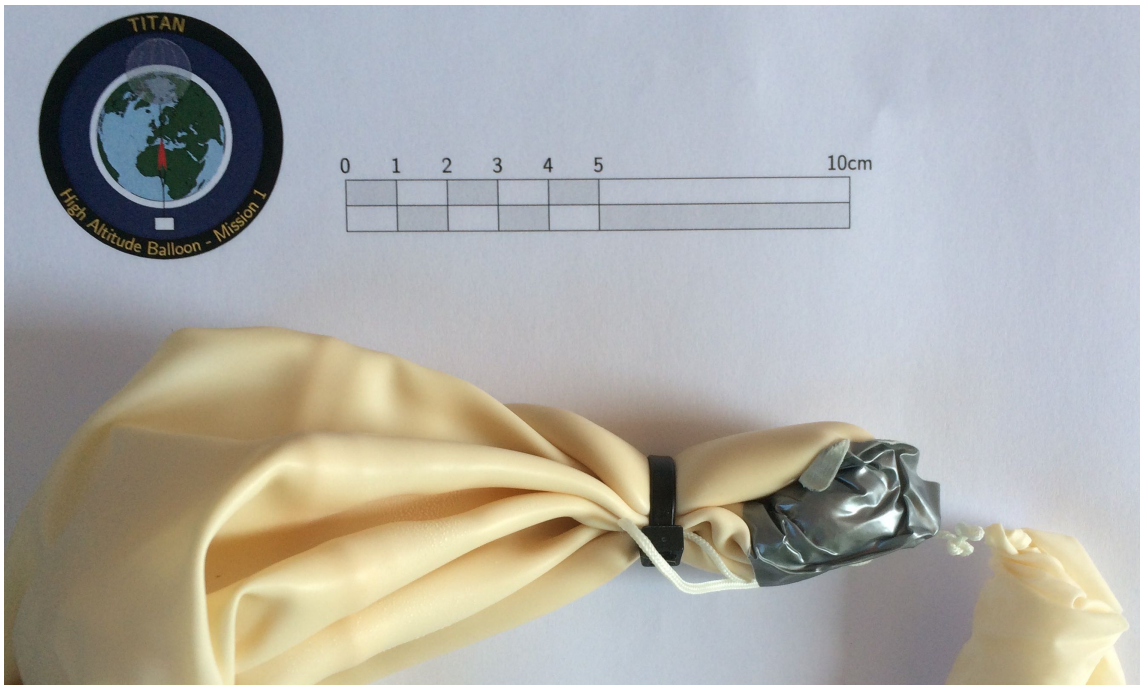


Figure 16: Sealed balloon and attached line to balloon latex part on the left. Image was taken post flight.

Fig. 16 shows the sealed balloon and the attached line to the balloon. Attaching the line and sealing the balloon is probably the most critical part during the whole launch operation.

7 Mission preparation

This final chapter discusses the critical final steps need before mission launch. At the end of this section launch checklists are presented.

7.1 Launch Site

The launch site was selected to be easy accessible, has no nearby obstacles (e.g. building or nearby trees) and lies in the western part of Sjælland so to see the Great Belt Bridge and the Island Fyn. The launch site is located at 11.396 402° longitude and 55.420 401° latitude. Fig. 17 shows a panoramic view of the launch site. The idea is to launch the on the grass area right to the driveway.

Some idea of traffic routes above the launch location can be obtained by looking at the aeronautical charts shown at https://skyvector.com.

Figure 17: View towards east of the launch site. The grass area to the right is from where Titan 1 was launched.

7.2 Launch Permit

This section describes in short what steps has to be done in Denmark in order to get flight permit. This section is not exhaustive and only gives an indication of the major steps involved.

In accordance to the national law, *BL 7-9 / Udg. 3 / 3. oktober 2014* point 2. a. Titan 1 HAB is categorised as **Let** (light). The same holds when considering the *SERA-forordninger* listed in the same document. Based on this an e-mail was sent to the Danish Transport and Construction Agency (http://www.trafikstyrelsen.dk/) asking for permission to launch. Which was kindly granted in a fast and straight forward way.

Further, they communicated directly with Naviair (http://www.naviair.dk/) and so the duty was reduced to communicate with Naviair priori to the launch.

It was tried to gain insurance cover for a flight in Denmark, unfortunately with no success.

7.3 Testing

A major point in reaching a successful mission is to test the flight vehicle thoroughly before a launch. The tests and results of some of the most important elements are discussed in this section.

During development all sensors and writing to SD-micro card was tested in all possible configuration. Especially disconnecting or connecting of any device during operation was investigated to make sure Watchdog is able to reset and restart the flight computer if it hangs.

A future major point, the power consumptions, was investigated in section 2.12 and thus made sure that the power source is designed with sufficient reserve.

Fig. 18 shows a tracking test while the payload box is transported with a car. Unfortunately most of the time the the transmitter was below the horizon.

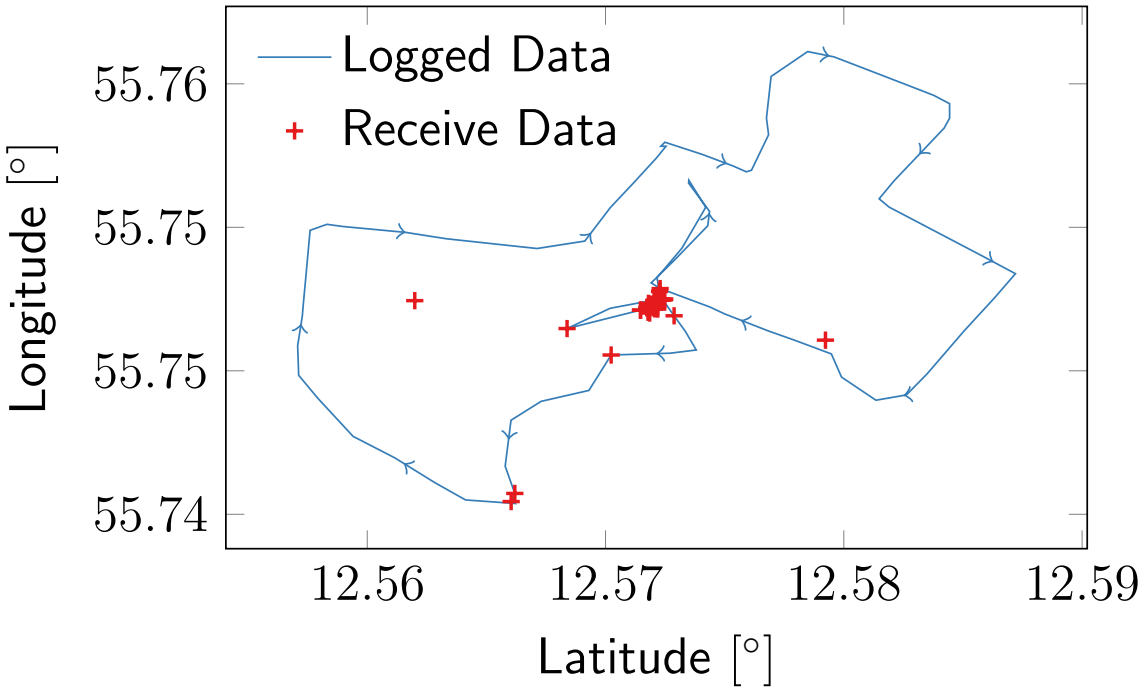


Figure 18: Test drive path of Titan 1 and received and decoded positions.

7.4 Fly Path prediction

One major point in a mission preparation is the prediction of the flight path. The flight path allows to evaluate if the mission can be conducted without endangering third parties. Further, it is an essential asset for flight traffic control. Finally, it increases success while tracking and recovering the HAB.

The flight path of a HAB is governed by the meteorological conditions in the area. Of special importance are the jet streams, which may drag the balloon considerable distances due its high wind velocities, see Fig. 19 showing jetstream forecasts over the northern hemisphere, found on http://www.netweather.tv/index.cgi?action=jetstream;sess=. Further, the atmopsheric pressure and wind are of importance. Forecast of the conditions are found on http://www.dmi.dk/vejr/til-lands/vejrkort/ and shown in Fig. 20.

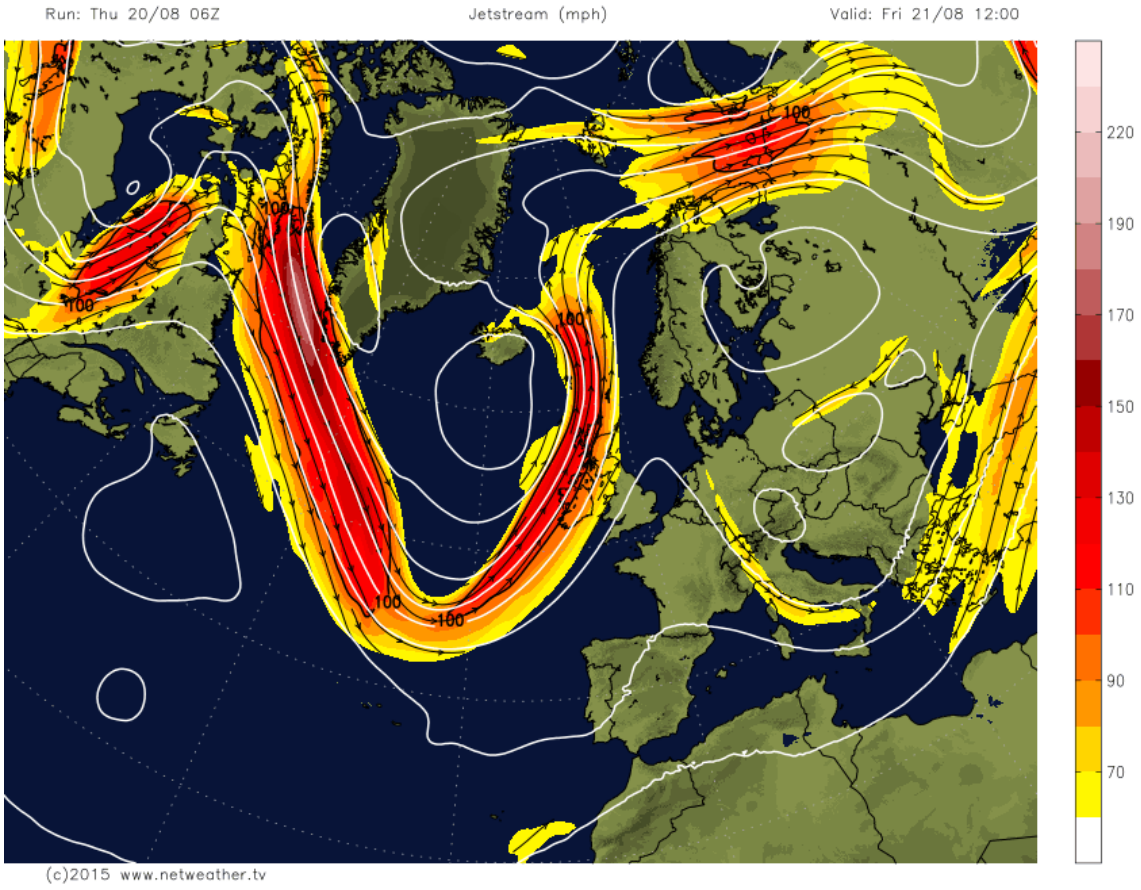


Figure 19: Jetstream forecast over northern hemisphere.

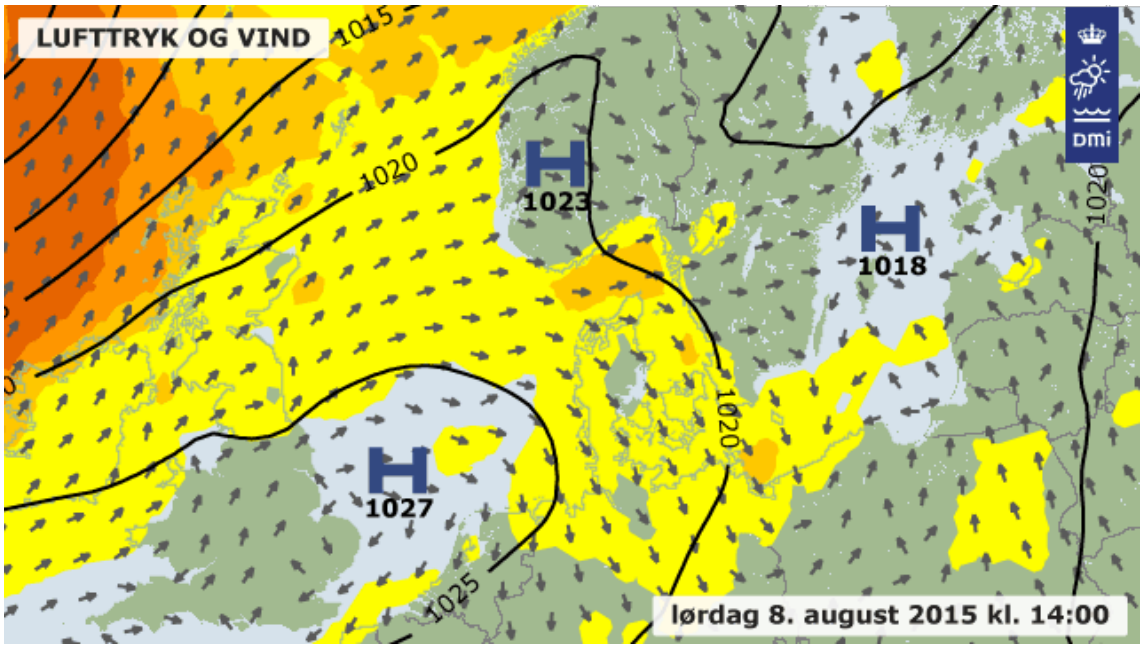


Figure 20: Forecast of atmospheric pressure, wind strength and direction.

The generation of the flight path prediction split into 3 distinct steps.

The first step is to look for possible launch windows up to 6 days ahead of a possible launch date. This is done by making forecast up to +160 h in 10 h steps. The individual flight path are shown in section Fig. 21. Based on these flight paths the best flight time and data is selected.

Once a possible flight time is found around 24 h before the launch a more refined analysis is carried out. This analysis calculates possible flights paths starting from −7 h up to +7 h from the target launch time. The paths are shown in section Fig. 22. This step is done in order to better understand how stable meteorological conditions around the launch time are. Kindly, recall discussion at the begin of this section.

The last step is done down to 24 h before the scheduled launch. To increase the reliability of the prediction in this step sensitivity studies are undertaken. Both the target ascent and descent velocity were increased and reduced by 10 %. Further, the peak altitude is reduced to 30 000 m.
In section Fig. 23, Titan's predicted flight path is shown together with all relevant information regarding **launch time**, penetration of **18 000ft level** and **predicted touchdown**.

Shortly before the flight this last step is repeated and checked for major deviations form the previous prediction.

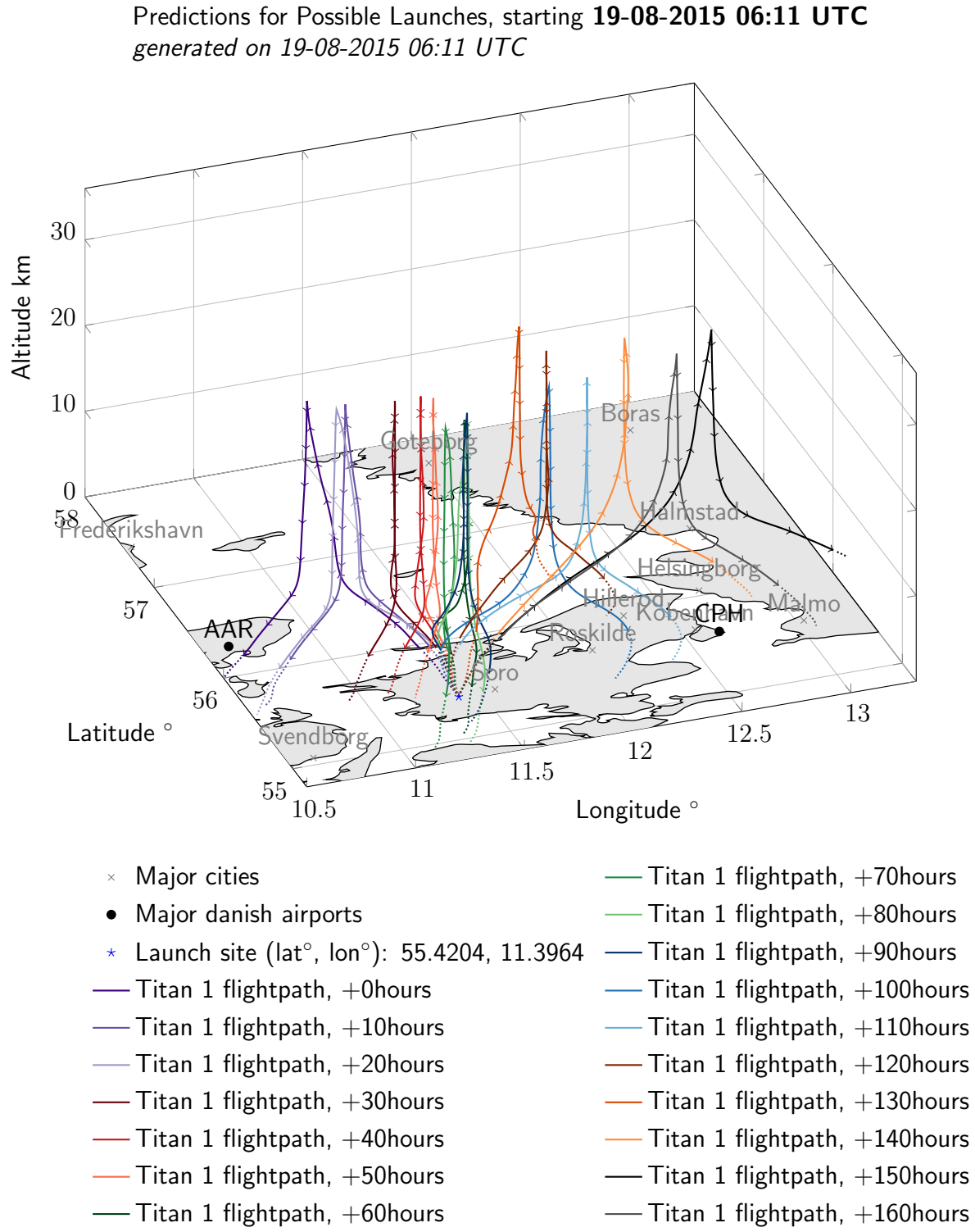


Figure 21: 160 h forecasts for identifying suitable launch date and time.

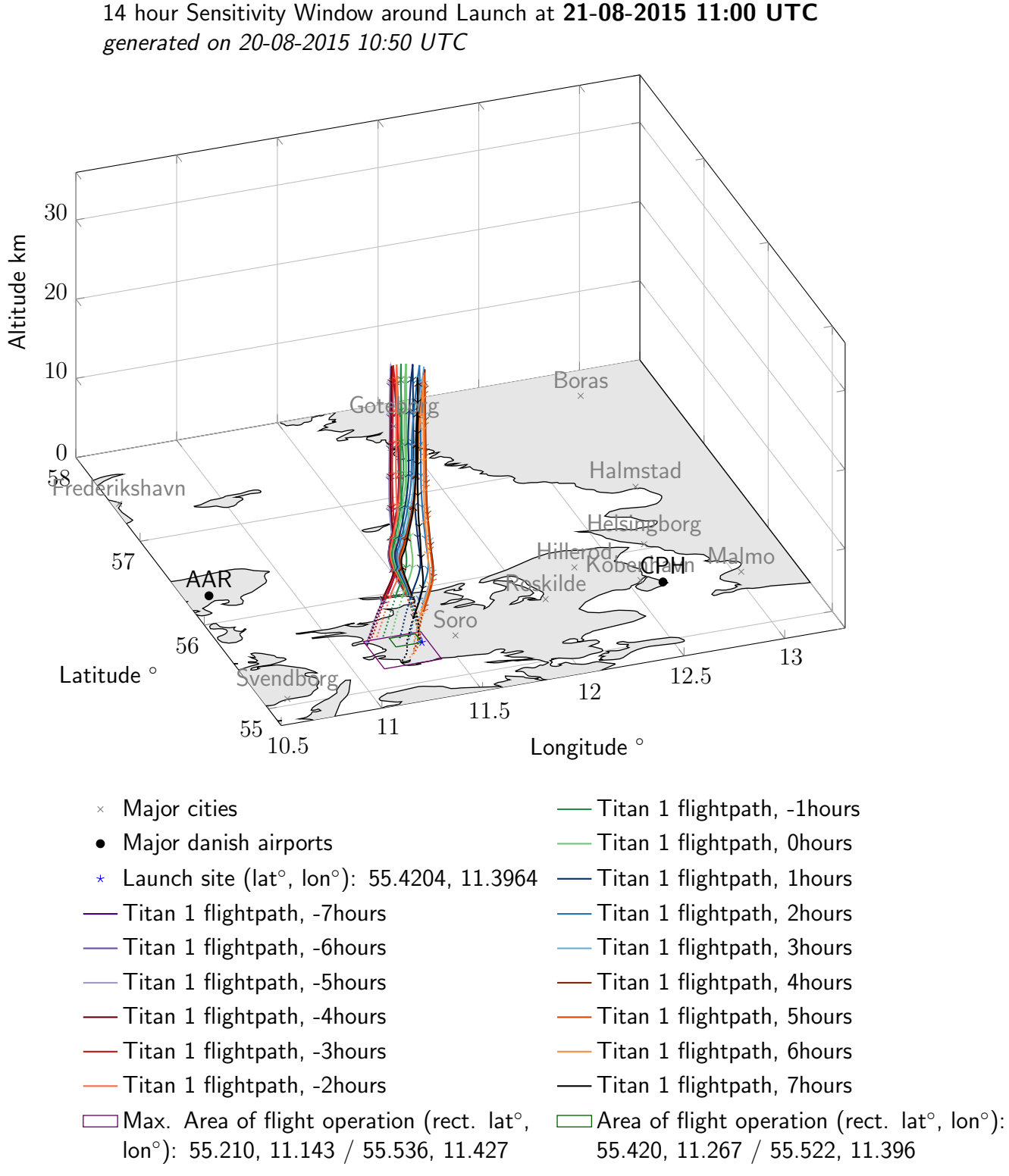
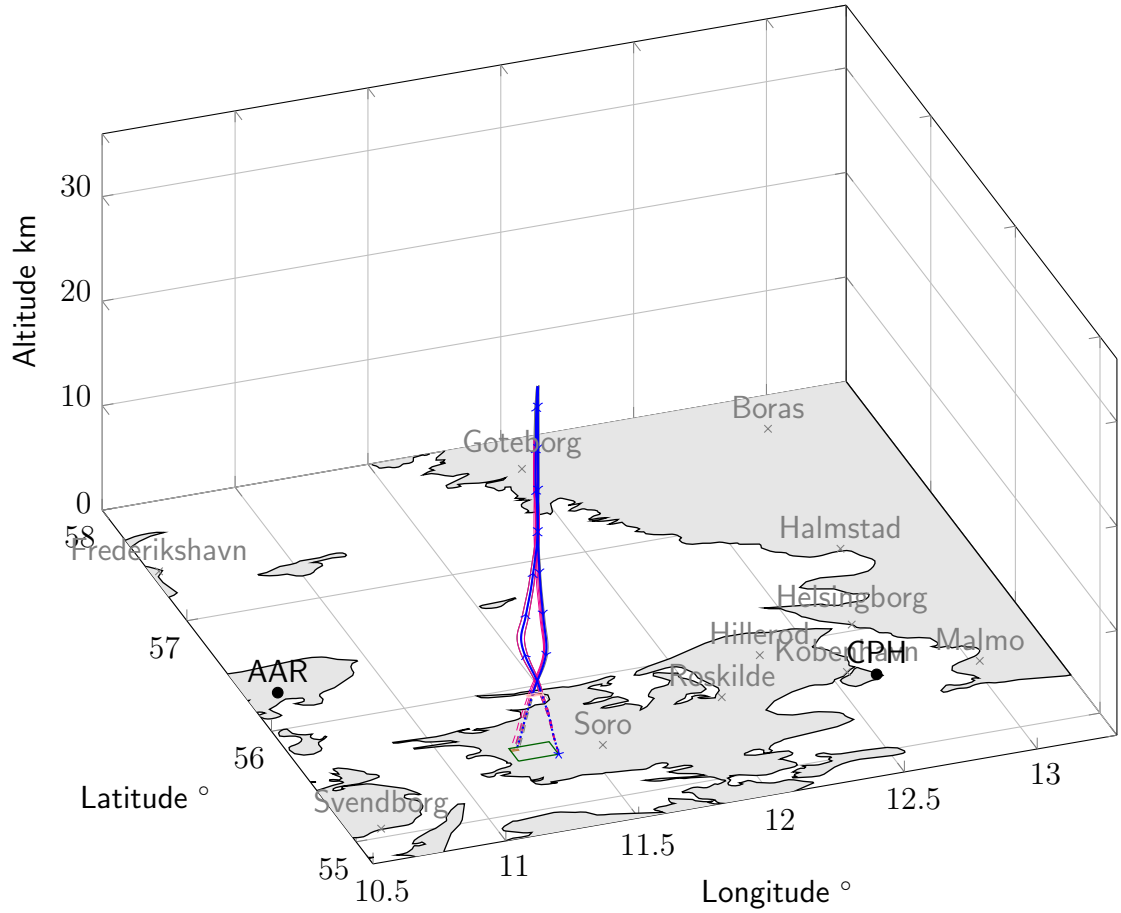


Figure 22: ±7 h simulations around launch time.

Prediction for launch on **Friday 21-Aug-2015 11:00:00 UTC**
generated on Thursday 20-Aug-2015 10:55:21 UTC



- Major cities
- Major danish airports
- Launch site (lat°, lon°): 55.4204, 11.3964
- Sensitivity, below 18000ft
- Sensitivity, above 18000ft
- Sensitivity peak altitude 30000m, below 18000ft
- Sensitivity peak altitude 30000m, above 18000ft
- Titan 1 flightpath, below 18000ft
- Titan 1 flightpath, above 18000ft
- 18000ft penetration zone (rect. lat°, lon°): 55.459, 11.292 / 55.480, 11.352
  Penetration time ascent: between 11:15:00 UTC and 11:18:20 UTC
  Penetration time descent: between 12:50:00 UTC and 13:32:30 UTC
- Estimated touchdown zone (rect. lat°, lon°): 55.507, 11.245 / 55.519, 11.274
  Touchdown time: between 13:04:53 UTC and 13:51:30 UTC
- Area of flight operation (rect. lat°, lon°): 55.420, 11.245 / 55.533, 11.396

Figure 23: Prediction for launch made ≈24 h before launch.

7.5 Pre Fly Day Checklist

☐ **Inform Flight control Launch -25 h**

- ☐ Run predictions and prepare Pre Flight Report.
- ☐ Send email to flight control. -24 h

☐ **Check Flight Computer, Sensors and Calibrate**

- ☐ Load calibration code flight on micro controller.
- ☐ Check all sensor data and get sensors calibrate values.
- ☐ Update calibration values in main core file.
- ☐ Load flight software on flight micro controller.
- ☐ Start up flight computer and check all sensors again.
- ☐ Format flight SD-micro and insert into flight computer.
- ☐ Check flight battery voltage and mark battery.
- ☐ Synchronise time.

☐ **Image Capture Preparation**

- ☐ Check settings
- ☐ Take trial images. Download images to computer and check them.
- ☐ Format SD-card.
- ☐ Load both batteries.
- ☐ Assemble camera and housing. Clean camera!
- ☐ Remember SD-card and fill with Dust-Off.
- ☐ Mount into payload box. Fix with tape. Check position.
- ☐ Synchronise time.

☐ **Packing Tools**

- ☐ Wire cutter, scissors, knife, pen, paper, torch, ...
- ☐ Gaffer tape
- ☐ Cable ties minimum 10 pieces.
- ☐ Payload box labels.
- ☐ Protecting sheet.
- ☐ Various tools.

☐ **Packing Payload**

- ☐ Payload box
- ☐ Check camera installed, battery ok and antenna ok.
- ☐ Payload box labels

☐ **Packing Balloon and Filling Equipment**

- ☐ Balloon
- ☐ Latex gloves
- ☐ Check tank and filling equipment.
- ☐ Tank, pressure reducer, hose, fill tube.
- ☐ Weight to check balloon fill state.

☐ **Packing Tracking Equipment**

- ☐ Track Antenna. Assembled Yagi antenna, FUNcube Dongle and USB cable.
- ☐ Flight Control Computer.
- ☐ Car power switch source.
- ☐ iPhone charge cable.

7.6 Pre Fly Checklist

- ☐ **Inform Flight control Launch** −60 min
  - ☐ Generate new prediction and check for deviations.
  - ☐ Call flight control at +45 and confirm launch and time.
- ☐ **Start Up Control Computer Launch** −50 min
  - ☐ Power up and power computer via car power switch source.
  - ☐ Connect iPhone via USB to car power switch.
  - ☐ Create local hotspot and connect computer to it.
  - ☐ Connect FUNcube Dongle via USB cable to lower USB.
  - ☐ Start DSP Radio, Fldigi and Flight Control Centre. Check frequency setting.
  - ☐ Start Flight Control Centre and check if working.
- ☐ **Start Up HAB Launch** −30 min
  - ☐ Lay out protecting sheet.
  - ☐ Connect battery to flight computer.
  - ☐ Wait till signal is received by flight control computer.
  - ☐ Check successful transmission.
  - ☐ Start image capture.
  - ☐ Check camcorder is operating.
  - ☐ Pack and close payload box.
  - ☐ Apply payload labels to seal payload box.
  - ☐ Lay out rigging lines and parachute and check for damage.
  - ☐ Check surrounding area to be free of obstacles.
- ☐ **Filling Balloon Launch** −20 min
  - ☐ Put gloves on.
  - ☐ Unpack and attach filling equipment to tanks. Remember check weights.
  - ☐ Open tank valve and check flow. Close tank valve.
  - ☐ Unpack balloon and put over filling tube. Fix balloon with cable ties and tape.
  - ☐ Start inflating balloon till nearly lift of check weights. Consider wind!
  - ☐ Connect rigging line with solid nodes.
  - ☐ Disconnect balloon from filling tube and close balloon end tight.

☐ **Launch Balloon Launch** 0 min

- ☐ Check aerial to not be bend.
- ☐ Check transmission and camera to record.
- ☐ Watch out for obstacles.
- ☐ Let it go.

7.7 Post Fly Checklist

☐ **On Site Recovery**

- ☐ Call flight control +45 and confirm touchdown or loss.
- ☐ Photograph payload box before touching.
- ☐ Recover payload box.
- ☐ Open payload box.
- ☐ Stop image capture.
- ☐ Disconnect battery.
- ☐ Close payload box.
- ☐ Power down flight control computer.

☐ **In Base**

- ☐ Copy logged data to computer.
- ☐ Copy images to computer.
- ☐ Copy Fldigi log.
- ☐ Backup data.
- ☐ Analyse data and prepare Flight Data Report.

A Flight Computer Software Source Code

A.1 Core File

Listing 3: File *titan_v01.ino* main core file of the flight software.

```
/*
 TITAN HAB flight computer software
 Modifications copyright 2015 by Emanuel Bombasaro

 The original code and subfiles is by:
   HABDuino Tracker
   http://www.habduino.org
   (c) Anthony Stirk M0UPU
   November 2014 Version 3.0.3
   Please visite for lates release and details: https://github.com/HABduino
    /HABduino

   Modifications done are:
   Make it work with Arduino Mega 2560 rev3
   All parts regarding additional sensors and logging to SD-card.
   Some modifications on formatting and variable naming.
   However, please start with looking into the original code first!

 This program is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU General Public License for more details.

 See <http://www.gnu.org/licenses/>.

 This code is suitable for the Arduino MEGA 2560 rev3.
 Pin connection as outlined below.

 HABduino, mounat as it is.
 NB. I2C is connected as follows:
   connect SCL to analog 21
   connect SDA to analog 20
 thus the pins on the HABduino are not correct.

 SD card attached to SPI bus as follows:
   MISO - pin 50
   MOSI - pin 51
   SLK - pin 52
   SS - pin 53

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
created 25-XII-2014 EMBO
edited  20- II-2015 EMBO
edited  03- III-2015 EMBO fixed sd write, no in the delay loop seems to
    work better.
edited  16- III-2015 EMBO reduced error value for sensors
```

```
50 edited  18- III-2015 EMBO gyro, acc and mag calibration ands sensor
       redadings
51 edited  24- III-2015 EMBO mag calibration now advance
52 */
53
54 // include ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
55 //HABduino
56 #include <util/crc16.h> // crc check sum
57 #include <avr/io.h>
58 #include <avr/interrupt.h>
59 #include <avr/pgmspace.h>
60 #include <avr/wdt.h> // watch dog
61 #include <Wire.h> // including I2C
62 #include <OneWire.h>
63 #include <DallasTemperature.h>
64 #include <SoftwareSerial.h>
65 #include "ax25modem.h"
66 // sensors
67 #include "Sensor.h"      // info on sensors, variables, strucutres
68 #include "TSL2561.h"     // light sensor
69 #include "HTU21DF.h"     // temp & humidity
70 #include "MPL3115A2.h"   // altimeter
71 #include "MCP9808.h"     // temperrature precision
72 #include "BMP085_U.h"    // pressure sensor IMU
73 // IMU 10 DOF
74 #include "L3GD20.h"      // gyro sensors
75 #include "LSM303_U.h"    // acc and mag sensor
76 /*
77 coordinates as indicated on sensor
78   gyro x, y, z rotation deg/s
79   acc  x, y, z accelaration m/s^2 (negative in direction of axis)
80   mag  x, y, z gauss
81   -------------------------------
82   roll postive x axis
83   pitch positive y axis
84   heading in positiv x direction
85 */
86 // sd card
87 #include <SPI.h>
88 #include <SD.h>
89
90 // configurations ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
91 // define constants
92 #define ONE_SECOND F_CPU / 1024 / 16  // define one second
93 #define MTX2_FREQ 434.485             // format 434.XXX
94 #define POWERSAVING                   // GPS power saving mode
95
96 // some help constants
97 #define BAUD_RATE      (1200)
98 #define TABLE_SIZE     (512)
99 #define PREAMBLE_BYTES (50)
100 #define REST_BYTES     (5)
101
102 #define PLAYBACK_RATE    (F_CPU / 256)
103 #define SAMPLES_PER_BAUD (PLAYBACK_RATE / BAUD_RATE)
104 #define PHASE_DELTA_1200 (((TABLE_SIZE * 1200L) << 7) / PLAYBACK_RATE)
105 #define PHASE_DELTA_2200 (((TABLE_SIZE * 2200L) << 7) / PLAYBACK_RATE)
106 #define PHASE_DELTA_XOR  (PHASE_DELTA_1200 ^ PHASE_DELTA_2200)
```

```
107
108 // RTTY HABduino
109 #define ASCII 7           // ASCII 7 or 8 bit
110 #define STOPBITS 2        // Either 1 or 2 stop bits
111 #define TXDELAY 0         // Delay between sentence TX's
112 #define RTTY_BAUD 50      // RTTY Baud rate (Recommended = 50)
113 #define MTX2_SHIFT 425    // shift rate
114 #define MTX2_OFFSET 0     // 0-100 Slightly adjusts the frequency by
       increasing the PWM
115
116 // pins ###############################################################
117 #define LED_WARN 12       // warning LED, red (hardware set)
118 #define LED_OK 13         // ok LED, green (hardware set)
119
120 #define BATTERY_ADC A0    // battery voltage check 0-1023 coresponding to
       0-5V (hardware set)
121 #define GPS_ON 2          // GPS power on (hardware set)
122 #define ONE_WIRE_BUS 5    // BUS pin for temperature sensor on HABduino (
       hardware set)
123
124 #define HX1_ENABLE 6      // for addtional ATTY element
125 #define HX1_TXD 3         // TXD for addtional ATTY (hardware set)
126 #define MTX2_ENABLE 7     // MTX2 enable  (hardware set)
127 #define MTX2_TXD 11       // MTX2 TXD (hardware set)
128
129 // ####################################################################
130 // call sign
131 char callsign[9] = "TITAN_1";           // MAX 9 CHARACTERS!!
132 // limit of staying in pedestrian mode
133 int pedlim=2000; // meters
134 // logfile name
135 char logfilename[13] = "01datlog.dat";     // cmax 8 char befor dot!
136
137 // define errorstatus
138 int errorstatus=0;
139 /* Error Status Bit Level Field :
140  Bit 0 = GPS Error Condition Noted Switch to Max Performance Mode
141  Bit 1 = GPS Error Condition Noted Cold Boot GPS
142  Bit 2 = DS18B20 temp sensor status 0 = OK 1 = Fault
143  Bit 3 = Current Dynamic Model 0 = Flight 1 = Pedestrian
144  Bit 4 = PSM Status 0 = PSM On 1 = PSM Off
145  Bit 5 = Lock 0 = GPS Locked 1= Not Locked
146
147  Bit 6 = TSL2561 lumosity sensor status 0 = OK 1 = Fault
148  Bit 7 = HTU21DF temp humidity sensor status 0 = OK 1 = Fault
149  Bit 8 = MPL3115A2 pressure sensor status 0 = OK 1 = Fault
150  Bit 9 = MCP9808 temperatur precision sensor status 0 = OK 1 = Fault
151  Bit 10= BMP085 pressure sensor status 0 = OK 1 = Fault
152  Bit 11= Gyro sensor status 0 = OK 1 = Fault
153  Bit 12= Acc sensor status 0 = OK 1 = Fault
154  Bit 13= Mag sensor status 0 = OK 1 = Fault
155  Bit 14= logging status 0 = OK 1 = Fault
156
157  So error 8 means the everything is fine just the GPS is in pedestrian mode
       .
158  Below pedlim meters the code puts the GPS in the more accurate pedestrian
       mode.
159  Above pedlim meters it switches to dynamic model 6 i.e flight mode and
```

```
   turns the LED's off for additional power saving.
160 So as an example error code 40 = 1010 0000 0000 means GPS not locked and
   in pedestrian mode.
161 */
162
163 // define vaiables
164 volatile int txstatus=1;
165 volatile int txstringlength=0;
166 volatile char txc;
167 volatile int txi;
168 volatile int txj;
169 volatile int count=1;
170 volatile boolean lockvariables = 0;
171 volatile static uint8_t *_txbuf = 0;
172 volatile static uint8_t  _txlen = 0;
173
174 char txstring[100]; // string values are stored in for transmission
175 char tmpS[2]; // temporay string for hour, minute and second logging
176 uint8_t buf[60];
177 uint8_t lock =0, sats = 0, hour = 0, minute = 0, second = 0;
178 uint8_t oldhour = 0, oldminute = 0, oldsecond = 0;
179 int GPSerror = 0,navmode = 0,psm_status = 0,lat_int=0,lon_int=0,
   temperature1=0;
180 int32_t lat = 0, lon = 0, alt = 0, maxalt = 0, lat_dec = 0, lon_dec =0 ,
   tslf=0;
181 unsigned long currentMillis;
182 long previousMillis = 0;
183 int batteryadc_v,battvaverage=0,aprstxstatus=0;
184 int32_t battvsmooth[5] ;
185 int aprs_tx_status = 0, aprs_attempts = 0;
186 unsigned long startTime;
187 char comment[3]={' ', ' ', '\0'};
188 static const uint8_t PROGMEM _sine_table[] = {
189 #include "sine_table.h"
190 }; // sinus wave
191 // sensor variables
192 uint32_t lum1=0, lux1=0;
193 uint16_t ir1=0, full1=0;
194 float temperature3=0, temperature4=0, humidity1=0, pressure1=0, alt1=0,
   temperature5=0, pressure2=0, alt2=0, temperature6=0;
195 float gyroX=0, gyroY=0, gyroZ=0, accX=0, accY=0, accZ=0, magX=0, magY=0,
   magZ=0, roll=0, pitch=0, head=0;
196 float accXnorm, accYnorm, magXnorm, magYnorm, magZnorm, magXcomp, magYcomp;
197 File dataFile; // log data file
198
199 // start serials and sensors
200 SoftwareSerial MTX2_EN(LED_WARN, MTX2_ENABLE); // RX, TX serial to
   comunitate with the MTX2
201 OneWire oneWire(ONE_WIRE_BUS);
202 DallasTemperature sensors(&oneWire);
203 TSL2561 tsl(TSL2561_ADDR_FLOAT); // the ADDR pin float (addr 0x39), light
   sensor
204 HTU21DF htu = HTU21DF();  // temp humidity senosor
205 MPL3115A2 baro = MPL3115A2();
206 MCP9808 pretempsen = MCP9808(); // Create the MCP9808 temperature sensor
   object
207 BMP085_Unified bmp = BMP085_Unified(10085); // pressuer sensor IMU
208 L3GD20 gyro; // start gyro
```

```
209 LSM303_Accel_Unified accel = LSM303_Accel_Unified(54321); // start acc
      sensor
210 LSM303_Mag_Unified mag = LSM303_Mag_Unified(12345); // start mag sensor
211
212 // calib gyro offset 0 is no offeset
213 const float gyroXoff=-2.54, gyroYoff=0.36, gyroZoff=-0.33;
214 // calibrate mag
215 const float magEC[3] = {10.1555, -0.271402, -26.0203};
216 const float magET[3][3] = {{0.965256, -0.00477968, 0.0596485},
      {-0.00477968, 0.932893, 0.00230931}, {0.0596485, 0.00230931, 0.897073}};
217 //
218 const int chipSelect = 53;  // sd card chip select (CS) pin, on Mega 2560
      pin 53
219
220 // main setup function ++++++++++++++++++++++++++++++++++++++++++++++++++
221 void setup()
222 {
223 wdt_enable (WDTO_8S); // watchdog ....................................
224   /* rest times
225     WDTO_15MS
226     WDTO_30MS
227     WDTO_60MS
228     WDTO_120MS
229     WDTO_250MS
230     WDTO_500MS
231     WDTO_1S
232     WDTO_2S
233     WDTO_4S
234     WDTO_8S
235  */
236   // set pin modes
237   pinMode(MTX2_TXD, OUTPUT);
238   pinMode(LED_WARN, OUTPUT);
239   pinMode(HX1_ENABLE, OUTPUT);
240   pinMode(LED_OK,OUTPUT);
241   pinMode(MTX2_ENABLE, OUTPUT);
242   pinMode(GPS_ON, OUTPUT);
243   pinMode(BATTERY_ADC, INPUT);
244
245   // setups
246   blinkled(6);
247   setMTX2Frequency();
248   Serial.begin(9600);
249   digitalWrite(MTX2_ENABLE,HIGH);
250   blinkled(5);
251   digitalWrite(GPS_ON,HIGH);
252   blinkled(4);
253   resetGPS();
254   blinkled(3);
255   // set timer for TXD
256   TCCR1B = TCCR1B & 0b11111000 | 0x01; // Sets fast PWM on pin 11 12
      controled by timer1
257 #ifdef APRS
258   ax25_init(); // initialize APRS
259 #endif
260   blinkled(2);
261   setupGPS();
262   blinkled(1);
```

```
263   initialise_interrupt();
264   // initialise sensors
265   sensors.begin(); // begin temp sensor
266   //tempsensors=sensors.getDeviceCount(); // we know its only 1
267   // ligth sensor settings and check if senosrs are on
268   // Set or unset bit 6 indicating the light sensor is faulty
269   if(!tsl.begin()) {
270     tsl.setGain(TSL2561_GAIN_0X); // set no gain (for bright situtations)
271     tsl.setTiming(TSL2561_INTEGRATIONTIME_101MS); // medium resolution and
      speed
272     errorstatus |=(1 << 6);}
273   else {errorstatus &= ~(1 << 6);}
274   // Set or unset bit 7 indicating the temperature and humidity sensor is
      faulty
275   if(!htu.begin()) {errorstatus |=(1 << 7);} else {errorstatus &= ~(1 << 7)
      ;}
276   // Set or unset bit 8 indicating the altimeter and presure sensor is
      faulty
277   if(!baro.begin()) {errorstatus |=(1 << 8);} else {errorstatus &= ~(1 <<
      8);}
278   // Set or unset bit 9 indicating the precision temperature sensor is
      faulty
279   if(!pretempsen.begin()) {errorstatus |=(1 << 9);} else {errorstatus &=
      ~(1 << 9);}
280   // Set or unset bit 10 indicating the pressuer IMU sensor is faulty
281   if(!bmp.begin()) {errorstatus |=(1 << 10);} else {errorstatus &= ~(1 <<
      10);}
282   // Set or unset bit 11 indicating the gyro IMU sensor is faulty
283   if (!gyro.begin(gyro.L3DS20_RANGE_500DPS)) {errorstatus |=(1 << 11);}
      else {errorstatus &= ~(1 << 11);}
284   // Set or unset bit 12 indicating the accel IMU sensor is faulty
285   if (!accel.begin()) {errorstatus |=(1 << 12);} else {errorstatus &= ~(1
      << 12);}
286   // Set or unset bit 13 indicating the mag IMU sensor is faulty
287   mag.enableAutoRange(true); // Enable auto-gain mag sensor
288   if (!mag.begin()) {errorstatus |=(1 << 13);} else {errorstatus &= ~(1 <<
      13);}
289   // sd card
290   pinMode(chipSelect, OUTPUT);
291   // see if the card is present and can be initialized else writ erro in
      bit 13
292   if (!SD.begin(chipSelect)){errorstatus |=(1 << 14);} else {
293     // open (create) file and close it, as check
294     dataFile = SD.open(logfilename, FILE_WRITE);
295     dataFile.close();
296     errorstatus &= ~(1 << 14);
297     }
298 wdt_reset(); // ..........................................................
299 }
300 // end setup function
301
302 // begin main loop ##########################################################
303 void loop()    {
304 wdt_reset(); // ..........................................................
305   oldhour=hour;
306   oldminute=minute;
307   oldsecond=second;
308   gps_check_nav();
```

```
309   // Set bit 5 (Lock 0 = GPS Locked 1= Not Locked)
310   if(lock!=3) {errorstatus |=(1 << 5);}else{errorstatus &= ~(1 << 5);}
311   checkDynamicModel();
312 wdt_reset(); // ..........................................................
313 #ifdef APRS
314   if(sats>=4){
315     if (aprs_tx_status==0)
316     {
317       startTime=millis();
318       aprs_tx_status=1;
319     }
320     if(millis() - startTime > (APRS_TX_INTERVAL*60000)) {
321       aprs_tx_status=0;
322       send_APRS();
323       aprs_attempts++;
324     }
325   }
326 #endif
327 wdt_reset(); // ..........................................................
328 #ifdef POWERSAVING
329   if((lock==3) && (psm_status==0) && (sats>=5) &&((errorstatus & (1 << 0))
      ==0)&&((errorstatus & (1 << 1))==0)) // Check we aren't in an error
      condition
330   {
331     setGPS_PowerSaveMode();
332     wait(1000);
333     psm_status=1;
334     errorstatus &= ~(1 << 4); // Set Bit 4 Indicating PSM is on
335   }
336 #endif
337   if(!lockvariables) {
338
339     prepare_data();
340     if(alt>maxalt && sats >= 4)
341     {
342       maxalt=alt;
343     }
344   }
345   if((oldhour==hour&&oldminute==minute&&oldsecond==second)||sats<=4) {
346     tslf++;
347   }
348   else
349   {
350     tslf=0;
351     errorstatus &= ~(1 << 0); // Unset bit 0 (Clear GPS Error Condition
      Noted Switch to Max Performance Mode)
352     errorstatus &= ~(1 << 1); // Unset bit 1 (Clear GPS Error Condition
      Noted Cold Boot GPS)
353   }
354 wdt_reset(); // ..........................................................
355   if((tslf>10 && ((errorstatus & (1 << 0))==0)&&((errorstatus & (1 << 1))
      ==0))) {
356     setupGPS();
357     wait(125);
358     setGps_MaxPerformanceMode();
359     wait(125);
360     errorstatus |=(1 << 0); // Set Bit 1 (GPS Error Condition Noted Switch
      to Max Performance Mode)
```

```
361     psm_status=0;
362     errorstatus |=(1 << 4); // Set Bit 4 (Indicate PSM is disabled)
363   }
364   if(tslf>100 && ((errorstatus & (1 << 0))==1)&&((errorstatus & (1 << 1))
       ==0)) {
365     errorstatus |=(1 << 0); // Unset Bit 0 we've already tried that didn't
       work
366     errorstatus |=(1 << 1); // Set bit 1 indicating we are cold booting the
       GPS
367     Serial.flush();
368     resetGPS();
369     wait(125);
370     setupGPS();
371   }
372 wdt_reset(); // ..............................................................
373 }
374 // end main loop ############################################################
375
376 // start initialise_interrupt function ++++++++++++++++++++++++++++++++++++
377 void initialise_interrupt()
378 {
379 wdt_reset(); // ..............................................................
380   // initialize Timer5 so we cant use pin 46, 45, 44. timer is a 16-bit
       timer
381   cli();          // disable global interrupts
382   TCCR5A = 0;     // set entire TCCR5A register to 0
383   TCCR5B = 0;     // same for TCCR5B
384   OCR5A = F_CPU / 1024 / RTTY_BAUD - 1;  // set compare match register to
       desired timer count:
385   TCCR5B |= (1 << WGM12);   // turn on CTC mode:
386   // Set CS10 and CS12 bits for:
387   TCCR5B |= (1 << CS10);  // see in combination with below
388   TCCR5B |= (1 << CS12);  // 1024 prescaler (C12 CS11 CS10 set to 1 0 1)
389   // enable timer compare interrupt:
390   TIMSK5 |= (1 << OCIE5A); // Interrupt mask register (TIMSKx)
391   sei();          // enable global interrupts
392 }
393 // end initialise_interrupt function
394
395 // start ax25_init function +++++++++++++++++++++++++++++++++++++++++++++++
396 void ax25_init(void)
397 {
398 wdt_reset(); // ..............................................................
399   /* Fast PWM mode, non-inverting output on OC2A */
400   //TCCR2A = _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
401   //TCCR2B = _BV(CS20);
402   TCCR3B = TCCR3B & 0b11111000 | 0x01; // Sets fast PWM on pin 2 3 5
       controled by timer3
403   pinMode(HX1_TXD, OUTPUT);
404 }
405 // end ax25_init function
406
407 // start prepare data function ++++++++++++++++++++++++++++++++++++++++++++
408 void prepare_data() //
409 {
410 wdt_reset(); // ..............................................................
411   if(aprstxstatus==0)
412   {
```

```
413     // get flight computer temp
414     sensors.requestTemperatures();
415     temperature1=sensors.getTempCByIndex(0);
416     // Set or unset bit 2 indicating the temp sensor is faulty
417     if(temperature1==-127) {errorstatus |=(1 << 2);}else{errorstatus &= ~(1
        << 2);}
418   }
419   // gps set and get position
420   gps_check_lock();
421   gps_get_position();
422   gps_get_time();
423 wdt_reset(); // ..........................................................
424   // battery voltage
425   batteryadc_v=analogRead(BATTERY_ADC)*4.8;
426   battvsmooth[4] = battvsmooth[3];
427   battvsmooth[3] = battvsmooth[2];
428   battvsmooth[2] = battvsmooth[1];
429   battvsmooth[1] = battvsmooth[0];
430   battvsmooth[0] = batteryadc_v;
431   battvaverage = (battvsmooth[0]+battvsmooth[1]+ battvsmooth[2]+battvsmooth
        [3]+battvsmooth[4])/5;
432 wdt_reset(); // ..........................................................
433   // read light sensor if working, set bit 6 for error
434   if(tsl.begin()) {
435     lum1 = tsl.getFullLuminosity();
436     ir1 = lum1 >> 16;
437     full1 = lum1 & 0xFFFF; // visible is full - ir
438     lux1 = tsl.calculateLux(full1, ir1);
439     errorstatus &= ~(1 << 6);
440     }
441   else {
442     lum1 = NAN;
443     ir1 = NAN;
444     full1 = NAN; // visible is full - ir
445     lux1 = NAN;
446     errorstatus |=(1 << 6);
447   }
448 wdt_reset(); // ..........................................................
449   // read temperature and humidity sensor if working, set bit 7 for error
450   if(htu.begin()) {
451     temperature3 = htu.readTemperature(); // in degree C
452     humidity1 =htu.readHumidity(); // in RH %
453     errorstatus &= ~(1 << 7);
454     }
455   else {
456     temperature3 = NAN;
457     humidity1 =NAN;
458     errorstatus |=(1 << 7);
459   }
460 wdt_reset(); // ..........................................................
461   // read pressure and altimeter sensor if working, set bit 8 for error
462   if(baro.begin()) {
463     pressure1 = baro.getPressure(); // in Pa
464     alt1 = baro.getAltitude(); // m
465     temperature4 = baro.getTemperature(); // in degree C
466     errorstatus &= ~(1 << 8);
467     }
468   else {
```

```
469     pressure1 = NAN; // in Pa
470     alt1 = NAN; // m
471     temperature4 = NAN; // in degree C
472     errorstatus |=(1 << 8);
473   }
474 wdt_reset(); // ..........................................................
475   // read precision temperature sensor if working, set bit 9 for error
476   if(pretempsen.begin()) {
477     temperature5 = pretempsen.readTempC(); // in degree C
478     errorstatus &= ~(1 << 9);
479     }
480   else {
481     temperature5 = NAN;
482     errorstatus |=(1 << 9);
483   }
484 wdt_reset(); // ..........................................................
485   // read BMP085 pressure sensor if working, set bit 10 for error
486   sensors_event_t bmp_event; // create sensor event
487   bmp.getEvent(&bmp_event);
488   if(bmp_event.pressure) {
489     pressure2 = bmp_event.pressure; // in hPa
490     pressure2 = pressure2*100; // in Pa
491     alt2 = bmp.pressureToAltitude(1013.26,bmp_event.pressure); // m
492     bmp.getTemperature(&temperature6); // in degree C
493     errorstatus &= ~(1 << 10);
494     }
495   else {
496     pressure2 = NAN; // in Pa
497     alt2 = NAN; // m
498     temperature6 = NAN; // in degree C
499     errorstatus |=(1 << 10);
500   }
501 wdt_reset(); // ..........................................................
502   // read gyro IMU sensor if working, set bit 11 for error
503   if(gyro.begin()) {
504     // read gyro data in deg/s 5 times and make average
505     gyro.read();gyroX=gyro.data.x-gyroXoff;gyroY=gyro.data.y-gyroYoff;gyroZ
      =gyro.data.z-gyroZoff; // 1
506     gyro.read();gyroX+=gyro.data.x-gyroXoff;gyroY+=gyro.data.y-gyroYoff;
     gyroZ+=gyro.data.z-gyroZoff; // 2
507     gyro.read();gyroX+=gyro.data.x-gyroXoff;gyroY+=gyro.data.y-gyroYoff;
     gyroZ+=gyro.data.z-gyroZoff; // 3
508     gyro.read();gyroX+=gyro.data.x-gyroXoff;gyroY+=gyro.data.y-gyroYoff;
     gyroZ+=gyro.data.z-gyroZoff; // 4
509     gyro.read();gyroX+=gyro.data.x-gyroXoff;gyroY+=gyro.data.y-gyroYoff;
     gyroZ+=gyro.data.z-gyroZoff; // 5
510     gyroX=-gyroX/5;gyroY=-gyroY/5;gyroZ=gyroZ/5; // average
511     errorstatus &= ~(1 << 11);
512     } else {
513     gyroX=NAN;
514     gyroY=NAN;
515     gyroZ=NAN;
516     errorstatus |=(1 << 11);
517   }
518 wdt_reset(); // ..........................................................
519   // read accel IMU sensor if working, set bit 12 for error
520   sensors_event_t acc_event; // create sensor event
521   accel.getEvent(&acc_event);
```

```
522   if(acc_event.acceleration.x) {
523     // read accel data in m/s^2 5 times and make average
524     accel.getEvent(&acc_event);accX=acc_event.acceleration.x;accY=acc_event
      .acceleration.y;accZ=acc_event.acceleration.z; // 1
525     accel.getEvent(&acc_event);accX+=acc_event.acceleration.x;accY+=
      acc_event.acceleration.y;accZ+=acc_event.acceleration.z; // 2
526     accel.getEvent(&acc_event);accX+=acc_event.acceleration.x;accY+=
      acc_event.acceleration.y;accZ+=acc_event.acceleration.z; // 3
527     accel.getEvent(&acc_event);accX+=acc_event.acceleration.x;accY+=
      acc_event.acceleration.y;accZ+=acc_event.acceleration.z; // 4
528     accel.getEvent(&acc_event);accX+=acc_event.acceleration.x;accY+=
      acc_event.acceleration.y;accZ+=acc_event.acceleration.z; // 5
529     accX=-accX/5;accY=-accY/5;accZ=-accZ/5; // average
530     errorstatus &= ~(1 << 12);
531     } else {
532     accX=NAN;
533     accY=NAN;
534     accZ=NAN;
535     errorstatus |=(1 << 12);
536   }
537 wdt_reset(); // ..............................................................
538   // read mag IMU sensor if working, set bit 13 for error
539   sensors_event_t mag_event; // create sensor event
540   mag.getEvent(&mag_event);
541   if(mag_event.magnetic.x) {
542     // read mag data in micro Tesla 5 times and make average, we correct
      the values for offesets
543     mag.getEvent(&mag_event);
544     magX=magET[0][0]*(mag_event.magnetic.x-magEC[0])+magET[0][1]*(mag_event
      .magnetic.y-magEC[1])+magET[0][2]*(mag_event.magnetic.z-magEC[2]);
545     magY=magET[1][0]*(mag_event.magnetic.x-magEC[0])+magET[1][1]*(mag_event
      .magnetic.y-magEC[1])+magET[1][2]*(mag_event.magnetic.z-magEC[2]);
546     magZ=magET[2][0]*(mag_event.magnetic.x-magEC[0])+magET[2][1]*(mag_event
      .magnetic.y-magEC[1])+magET[2][2]*(mag_event.magnetic.z-magEC[2]); // 1
547     mag.getEvent(&mag_event);
548     magX+=magET[0][0]*(mag_event.magnetic.x-magEC[0])+magET[0][1]*(
      mag_event.magnetic.y-magEC[1])+magET[0][2]*(mag_event.magnetic.z-magEC
      [2]);
549     magY+=magET[1][0]*(mag_event.magnetic.x-magEC[0])+magET[1][1]*(
      mag_event.magnetic.y-magEC[1])+magET[1][2]*(mag_event.magnetic.z-magEC
      [2]);
550     magZ+=magET[2][0]*(mag_event.magnetic.x-magEC[0])+magET[2][1]*(
      mag_event.magnetic.y-magEC[1])+magET[2][2]*(mag_event.magnetic.z-magEC
      [2]); // 2
551     mag.getEvent(&mag_event);
552     magX+=magET[0][0]*(mag_event.magnetic.x-magEC[0])+magET[0][1]*(
      mag_event.magnetic.y-magEC[1])+magET[0][2]*(mag_event.magnetic.z-magEC
      [2]);
553     magY+=magET[1][0]*(mag_event.magnetic.x-magEC[0])+magET[1][1]*(
      mag_event.magnetic.y-magEC[1])+magET[1][2]*(mag_event.magnetic.z-magEC
      [2]);
554     magZ+=magET[2][0]*(mag_event.magnetic.x-magEC[0])+magET[2][1]*(
      mag_event.magnetic.y-magEC[1])+magET[2][2]*(mag_event.magnetic.z-magEC
      [2]); // 3
555     mag.getEvent(&mag_event);
556     magX+=magET[0][0]*(mag_event.magnetic.x-magEC[0])+magET[0][1]*(
      mag_event.magnetic.y-magEC[1])+magET[0][2]*(mag_event.magnetic.z-magEC
      [2]);
```

```
557     magY+=magET[1][0]*(mag_event.magnetic.x-magEC[0])+magET[1][1]*(
        mag_event.magnetic.y-magEC[1])+magET[1][2]*(mag_event.magnetic.z-magEC
        [2]);
558     magZ+=magET[2][0]*(mag_event.magnetic.x-magEC[0])+magET[2][1]*(
        mag_event.magnetic.y-magEC[1])+magET[2][2]*(mag_event.magnetic.z-magEC
        [2]); // 4
559     mag.getEvent(&mag_event);
560     magX+=magET[0][0]*(mag_event.magnetic.x-magEC[0])+magET[0][1]*(
        mag_event.magnetic.y-magEC[1])+magET[0][2]*(mag_event.magnetic.z-magEC
        [2]);
561     magY+=magET[1][0]*(mag_event.magnetic.x-magEC[0])+magET[1][1]*(
        mag_event.magnetic.y-magEC[1])+magET[1][2]*(mag_event.magnetic.z-magEC
        [2]);
562     magZ+=magET[2][0]*(mag_event.magnetic.x-magEC[0])+magET[2][1]*(
        mag_event.magnetic.y-magEC[1])+magET[2][2]*(mag_event.magnetic.z-magEC
        [2]); // 5
563     magX=magX/5;magY=magY/5;magZ=magZ/5; // average
564     errorstatus &= ~(1 << 13);
565     } else {
566     magX=NAN;
567     magY=NAN;
568     magZ=NAN;
569     errorstatus |=(1 << 13);
570   }
571 wdt_reset(); // ..............................................................
572   // calculate roll, pitch and heading from acc and mag readings
573   // Normalize acceleration measurements so they range from 0 to 1
574   accXnorm = -accX/sqrt(accX*accX+accY*accY+accZ*accZ);
575   accYnorm = -accY/sqrt(accX*accX+accY*accY+accZ*accZ);
576   // calculate pitch and roll in rad EQ. 10
577   pitch = asin(-accXnorm);
578   roll = asin(accYnorm/cos(pitch));
579   // tilt compensated magnetic sensor measurements
580   magXnorm = magX/sqrt(magX*magX+magY*magY+magZ*magZ);
581   magYnorm = magY/sqrt(magX*magX+magY*magY+magZ*magZ);
582   magZnorm = magZ/sqrt(magX*magX+magY*magY+magZ*magZ);
583   // mag components EQ. 12
584   magXcomp = magXnorm*cos(pitch)+magZnorm*sin(pitch);
585   magYcomp = magXnorm*sin(roll)*sin(pitch)+magYnorm*cos(roll)-magZnorm*sin(
       roll)*cos(pitch);
586   // calculate heading EQ. 13
587   head = atan2(magYcomp,magXcomp)*180/PI;
588   if (head < 0) {head +=360;}
589   // transpose pitch and roll into degree
590   pitch *= 180/PI;
591   roll *= 180/PI;
592 wdt_reset(); // ..............................................................
593 }
594 // end prepare data function
595
596 // start writeToSD function ++++++++++++++++++++++++++++++++++++++++++++++++
597 void writeToSD() //
598 {
599 wdt_reset(); // ..............................................................
600   dataFile = SD.open(logfilename, FILE_WRITE); // if file does not exist it
        will be created
601   if (dataFile) {
602       // write transmited string
```

```
603       dataFile.print(callsign);dataFile.print(",");
604       dataFile.print(count);dataFile.print(",");
605       snprintf(tmpS,10,"%02d:%02d:%02d",hour,minute,second);
606       dataFile.print(tmpS);dataFile.print(",");
607       dataFile.print(lat < 0 ? "-" : "");
608       dataFile.print(lat_int);dataFile.print(".");
609       dataFile.print(lat_dec);dataFile.print(",");
610       dataFile.print(lon < 0 ? "-" : "");
611       dataFile.print(lon_int);dataFile.print(".");
612       dataFile.print(lon_dec);dataFile.print(",");
613       dataFile.print(maxalt);dataFile.print(",");
614       dataFile.print(sats);dataFile.print(",");
615       dataFile.print(temperature1);dataFile.print(","); // DS18B20
616       dataFile.print(battvaverage);dataFile.print(",");
617       dataFile.print(errorstatus);dataFile.print(",");
618       // write remaining sensor data
619       dataFile.print(lum1);dataFile.print(","); // TSL2561
620       dataFile.print(ir1);dataFile.print(","); // TSL2561
621       dataFile.print(full1);dataFile.print(","); // TSL2561
622       dataFile.print(lux1);dataFile.print(","); // TSL2561
623       dataFile.print(temperature3,2);dataFile.print(","); // HTU21DF
624       dataFile.print(humidity1,2);dataFile.print(","); // HTU21DF
625       dataFile.print(pressure1,0);dataFile.print(","); // MPL3115A2
626       dataFile.print(alt1,2);dataFile.print(","); // MPL3115A2
627       dataFile.print(temperature4,2);dataFile.print(","); // MPL3115A2
628       dataFile.print(temperature5,2);dataFile.print(","); // MCP9808
629       dataFile.print(pressure2,0);dataFile.print(","); // BMP085
630       dataFile.print(alt2,2);dataFile.print(","); // BMP085
631       dataFile.print(temperature6,2);dataFile.print(","); // BMP085
632       dataFile.print(gyroX,0);dataFile.print(","); // L3GD20
633       dataFile.print(gyroY,0);dataFile.print(","); // L3GD20
634       dataFile.print(gyroZ,0);dataFile.print(","); // L3GD20
635       dataFile.print(accX,2);dataFile.print(","); // LSM303 acc
636       dataFile.print(accY,2);dataFile.print(","); // LSM303 acc
637       dataFile.print(accZ,2);dataFile.print(","); // LSM303 acc
638       dataFile.print(magX,2);dataFile.print(","); // LSM303 mag
639       dataFile.print(magY,2);dataFile.print(","); // LSM303 mag
640       dataFile.print(magZ,2);dataFile.print(","); // LSM303 mag
641       dataFile.print(roll,2);dataFile.print(","); // LSM303 roll
642       dataFile.print(pitch,2);dataFile.print(","); // LSM303 pitch
643       dataFile.print(head,2);//dataFile.print(","); // LSM303 head
644       dataFile.print("\n"); // EOL
645       dataFile.close(); // close data file
646       errorstatus &= ~(1 << 14);
647   } else {errorstatus |=(1 << 14);}
648 wdt_reset(); // ..........................................................
649 }
650 // end writeToSD function
651
652 // start ISR timer5 function ++++++++++++++++++++++++++++++++++++++++++++++++
653 ISR(TIMER5_COMPA_vect)
654 {
655 wdt_reset(); // ..........................................................
656   if(alt>pedlim && sats >= 4)
657   {
658     digitalWrite(LED_WARN,LOW);
659     digitalWrite(LED_OK,LOW);
660   }
```

```
661   else
662   {
663     currentMillis = millis();
664     if(currentMillis - previousMillis > ONE_SECOND)
665     {
666       previousMillis = currentMillis;
667       if(errorstatus!=8)
668       {
669         digitalWrite(LED_WARN,!digitalRead(LED_WARN));
670         digitalWrite(LED_OK,LOW);
671       }
672       else
673       {
674         digitalWrite(LED_OK, !digitalRead(LED_OK));
675         digitalWrite(LED_WARN,LOW);
676       }
677     }
678   }
679  // switch
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
680   switch(txstatus) {
681 wdt_reset(); // ..............................................................
682   case 0: // This is the optional delay between transmissions.
683 wdt_reset(); // ..............................................................
684     txj++;
685     if(txj>(TXDELAY*RTTY_BAUD)) {
686       txj=0;
687       txstatus=1;
688     }
689     writeToSD(); // write to micro SD
690     break;
691   case 1: // Initialise transmission
692 wdt_reset(); // ..............................................................
693     if(alt>maxalt && sats >= 4)
694     {
695       maxalt=alt;
696     }
697     lockvariables=1;
698 #ifndef APRS
699     snprintf(txstring,100, "$$$$$$s,%i,%02d:%02d:%02d,%s%i.%06ld,%s%i.%06ld
    ,%ld,%d,%i,%i,%i",callsign,count, hour, minute, second,lat < 0 ? "-" : "
    ",lat_int,lat_dec,lon < 0 ? "-" : "",lon_int,lon_dec, maxalt,sats,
    temperature1,battvaverage,errorstatus);
700 #endif
701 #ifdef APRS
702     snprintf(txstring,100, "$$$$$$s,%i,%02d:%02d:%02d,%s%i.%06ld,%s%i.%06ld
    ,%ld,%d,%i,%i,%i,%i",callsign,count, hour, minute, second,lat < 0 ? "-"
    : "",lat_int,lat_dec,lon < 0 ? "-" : "",lon_int,lon_dec, maxalt,sats,
    temperature1,battvaverage,aprs_attempts,errorstatus);
703 #endif
704     crccat(txstring);
705     maxalt=0;
706     lockvariables=0;
707     txstringlength=strlen(txstring);
708     txstatus=2;
709     txj=0;
710     break;
711   case 2: // Grab a char and lets go transmit it.
```

```
712 wdt_reset(); // ..........................................................
713     if ( txj < txstringlength)
714     {
715       txc = txstring[txj];
716       txj++;
717       txstatus=3;
718       rtty_txbit (0); // Start Bit;
719       txi=0;
720     }
721     else
722     {
723       txstatus=0; // Should be finished
724       txj=0;
725       count++;
726     }
727     break;
728   case 3:
729 wdt_reset(); // ..........................................................
730     if(txi<ASCII)
731     {
732       txi++;
733       if (txc & 1) rtty_txbit(1);
734       else rtty_txbit(0);
735       txc = txc >> 1; // left shift one bit
736       break;
737     }
738     else
739     {
740       rtty_txbit (1); // Stop Bit
741       txstatus=4;
742       txi=0;
743       break;
744     }
745   case 4:
746 wdt_reset(); // ..........................................................
747     if(STOPBITS==2) // if two stop bits
748     {
749       rtty_txbit (1); // Stop Bit
750       txstatus=2;
751       break;
752     }
753     else
754     {
755       txstatus=2;
756       break;
757     }
758   }
759 }
760 // end ISR timer 1 function
761
762 // start ISR timer3 function ++++++++++++++++++++++++++++++++++++++++++++++++
763 ISR(TIMER3_OVF_vect)
764 {
765 wdt_reset(); // ..........................................................
766   static uint16_t phase  = 0;
767   static uint16_t step   = PHASE_DELTA_1200;
768   static uint16_t sample = 0;
769   static uint8_t rest    = PREAMBLE_BYTES + REST_BYTES;
```

```
770   static uint8_t byte;
771   static uint8_t bit     = 7;
772   static int8_t bc       = 0;
773   /* Update the PWM output */
774   OCR2B = pgm_read_byte(&_sine_table[(phase >> 7) & 0x1FF]);
775   phase += step;
776
777   if(++sample < SAMPLES_PER_BAUD) return;
778   sample = 0;
779
780   /* Zero-bit insertion */
781   if(bc == 5)
782   {
783     step ^= PHASE_DELTA_XOR;
784     bc = 0;
785     return;
786   }
787
788   /* Load the next byte */
789   if(++bit == 8)
790   {
791     bit = 0;
792
793     if(rest > REST_BYTES || !_txlen)
794     {
795       if(!--rest)
796       {
797         /* Disable radio and interrupt */
798         PORTD &= ~_BV(HX1_ENABLE); // Turn the HX1 Off
799         aprstxstatus=0;
800         TIMSK2 &= ~_BV(TOIE2);
801
802         /* Prepare state for next run */
803         phase = sample = 0;
804         step  = PHASE_DELTA_1200;
805         rest  = PREAMBLE_BYTES + REST_BYTES;
806         bit   = 7;
807         bc    = 0;
808         return;
809       }
810
811       /* Rest period, transmit ax.25 header */
812       byte = 0x7E;
813       bc = -1;
814     }
815     else
816     {
817       /* Read the next byte from memory */
818       byte = *(_txbuf++);
819       if(!--_txlen) rest = REST_BYTES + 2;
820       if(bc < 0) bc = 0;
821     }
822   }
823
824   /* Find the next bit */
825   if(byte & 1)
826   {
827     /* 1: Output frequency stays the same */
```

```
828     if(bc >= 0) bc++;
829   }
830   else
831   {
832     /* 0: Toggle the output frequency */
833     step ^= PHASE_DELTA_XOR;
834     if(bc >= 0) bc = 0;
835   }
836
837   byte >>= 1;
838 }
839 // end ISR timer2 function
840
841 // some more functions ######################################################
842
843 // start rtty_txbit function ++++++++++++++++++++++++++++++++++++++++++++++
844 void rtty_txbit (int bit)
845 {
846 wdt_reset(); // ..............................................................
847   if (bit) // bit is 1
848   {
849     analogWrite(MTX2_TXD, MTX2_OFFSET+((MTX2_SHIFT*1.8)/16)); // High
850   }
851   else // bit is 0
852   {
853     analogWrite(MTX2_TXD, MTX2_OFFSET); // Low
854   }
855 }
856 // end rtty_txbit function
857
858 // start blinkled function ++++++++++++++++++++++++++++++++++++++++++++++++
859 void blinkled(int blinks)
860 {
861 wdt_reset(); // ..............................................................
862   for(int blinkledx = 0; blinkledx <= blinks; blinkledx++) {
863     digitalWrite(LED_WARN,HIGH);
864     wait(100);
865     digitalWrite(LED_WARN,LOW);
866     wait(100);
867   }
868 }
869 // end blinkled function
870
871 // start wait function, we redefine so make sure it works like we want ++++
872 void wait(unsigned long delaytime) // Arduino Delay doesn't get CPU Speeds
      below 8Mhz
873 {
874   unsigned long _delaytime=millis();
875   while((_delaytime+delaytime)>=millis()){
876   }
877 }
878 // end wait function
879
880 // start crccat function ++++++++++++++++++++++++++++++++++++++++++++++++++
881 uint16_t crccat(char *msg)
882 {
883 wdt_reset(); // ..............................................................
884   uint16_t x;
```

```
885
886   while(*msg == '$') msg++;
887
888   for(x = 0xFFFF; *msg; msg++)
889     x = _crc_xmodem_update(x, *msg);
890
891   snprintf_P(msg, 8, PSTR("*%04X\n"), x);
892
893   return(x);
894 }
895 // end crccat function
896
897 // EOF ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

A.2 Function Files

Listing 4: File *mtx2_apresFun_v01.ino* containing function for controlling the radio transmission module.

```
1  /*
2   TITAN HAB flight computer software
3   Modifications copyright 2015 by Emanuel Bombasaro
4
5   This code is suitable for the Arduino Mega 2560 rev3.
6   Function file contianing all relvant functions regarding MTX2 and APRS
7
8  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
9  created 25-XII-2014 EMBO
10 edited  14- II-2015 EMBO
11 */
12
13 // MTX2 ###################################################################
14 // function setMTX2Frequency ++++++++++++++++++++++++++++++++++++++++++++
15 void setMTX2Frequency()
16 {
17   float _mtx2comp;
18   int _mtx2int;
19   long _mtx2fractional;
20   char _mtx2command[17];
21   MTX2_EN.begin(9600);
22   _mtx2comp=(MTX2_FREQ+0.0015)/6.5;
23   _mtx2int=_mtx2comp;
24   _mtx2fractional=float(((_mtx2comp-_mtx2int)+1)*524288);
25   snprintf(_mtx2command,17,"@PRG_%02X%06lX\r",_mtx2int-1, _mtx2fractional);
26   delay(100);
27   MTX2_EN.print(_mtx2command);
28   delay(50);
29   MTX2_EN.end();
30 }
31 // end setMTX2Frequency function
32
33 // APRS ###################################################################
34 // start send_APRS function ++++++++++++++++++++++++++++++++++++++++++++++
35 void send_APRS() {
36   ax25_init();
37   tx_aprs();
38 }
```

```
39 //end send_APRS function
40
41 // start tx_aprs function +++++++++++++++++++++++++++++++++++++++++++++++++
42 void tx_aprs()
43 {
44   aprstxstatus=1;
45   PORTD |= _BV(HX1_ENABLE); // Same as digitalWrite(HX1_ENABLE, HIGH); but
       more efficient
46   char slat[5];
47   char slng[5];
48   char stlm[9];
49   static uint16_t seq = 0;
50   double aprs_lat, aprs_lon;
51
52   /* Convert the UBLOX-style coordinates to
53      * the APRS compressed format */
54   aprs_lat = 900000000 - lat;
55   aprs_lat = aprs_lat / 26 - aprs_lat / 2710 + aprs_lat / 15384615;
56   aprs_lon = 900000000 + lon / 2;
57   aprs_lon = aprs_lon / 26 - aprs_lon / 2710 + aprs_lon / 15384615;
58   int32_t aprs_alt = alt * 32808 / 10000;
59
60
61   /* Construct the compressed telemetry format */
62   ax25_base91enc(stlm + 0, 2, seq);
63     ax25_frame(
64     APRS_CALLSIGN, APRS_SSID,
65     "APRS", 0,
66     //0, 0, 0, 0,
67     "WIDE1", 1, "WIDE2",1,
68     //"WIDE2", 1,
69     "!/%s%sO   /A=%06ld|%s|%s/%s,%d,%i,%i'C",
70     ax25_base91enc(slat, 4, aprs_lat),
71     ax25_base91enc(slng, 4, aprs_lon),
72     aprs_alt, stlm, comment,APRS_CALLSIGN, count, errorstatus,temperature1)
       ;
73   seq++;
74 }
75 // end tx_aprs function
76
77 // start ax25_frame function +++++++++++++++++++++++++++++++++++++++++++++++
78 void ax25_frame(char *scallsign, char sssid, char *dcallsign, char dssid,
79 char *path1, char ttl1, char *path2, char ttl2, char *data, ...)
80 {
81   static uint8_t frame[100];
82   uint8_t *s;
83   uint16_t x;
84   va_list va;
85
86   va_start(va, data);
87
88   /* Pause while there is still data transmitting */
89   while(_txlen);
90
91   /* Write in the callsigns and paths */
92   s = _ax25_callsign(frame, dcallsign, dssid);
93   s = _ax25_callsign(s, scallsign, sssid);
94   if(path1) s = _ax25_callsign(s, path1, ttl1);
```

```
95    if(path2) s = _ax25_callsign(s, path2, ttl2);
96
97    /* Mark the end of the callsigns */
98    s[-1] |= 1;
99
100   *(s++) = 0x03; /* Control, 0x03 = APRS-UI frame */
101   *(s++) = 0xF0; /* Protocol ID: 0xF0 = no layer 3 data */
102
103   vsnprintf((char *) s, 100 - (s - frame) - 2, data, va);
104   va_end(va);
105
106   /* Calculate and append the checksum */
107   for(x = 0xFFFF, s = frame; *s; s++)
108     x = _crc_ccitt_update(x, *s);
109
110   *(s++) = ~(x & 0xFF);
111   *(s++) = ~((x >> 8) & 0xFF);
112
113   /* Point the interrupt at the data to be transmit */
114   _txbuf = frame;
115   _txlen = s - frame;
116
117   /* Enable the timer and key the radio */
118   TIMSK2 |= _BV(TOIE2);
119   //PORTA |= TXENABLE;
120 }
121 // end ax25_frame function
122
123 // start _ax25_callsign function ++++++++++++++++++++++++++++++++++++++++++++
124 static uint8_t *_ax25_callsign(uint8_t *s, char *callsign, char ssid)
125 {
126   char i;
127   for(i = 0; i < 6; i++)
128   {
129     if(*callsign) *(s++) = *(callsign++) << 1;
130     else *(s++) = ' ' << 1;
131   }
132   *(s++) = ('0' + ssid) << 1;
133   return(s);
134 }
135 // end _ax25_callsign function
136
137 // start ax25_base91enc function ++++++++++++++++++++++++++++++++++++++++++++
138 char *ax25_base91enc(char *s, uint8_t n, uint32_t v)
139 {
140   /* Creates a Base-91 representation of the value in v in the string */
141   /* pointed to by s, n-characters long. String length should be n+1. */
142
143   for(s += n, *s = '\0'; n; n--)
144   {
145     *(--s) = v % 91 + 33;
146     v /= 91;
147   }
148
149   return(s);
150 }
151 // end ax25_base91enc function
152
```

```
153 // EOF ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Listing 5: File *gpsFun_v01.ino* containing function for operating the GPS module.

```
1  /*
2   TITAN HAB flight computer software
3   Modifications copyright 2015 by Emanuel Bombasaro
4
5   This code is suitable for the Arduino Mega 2560 rev3.
6   Function file contianing all relvant functions regarding GPS
7
8  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
9  created 25-XII-2014 EMBO
10 edited  22-  I-2015 EMBO
11 */
12
13 /*
14 Dynamic platform model:
15 0: portable
16 2: stationary
17 3: pedestrian (used)
18 4: automotive
19 5: sea
20 6: airborne with <1g Acceleration (used)
21 7: airborne with <2g Acceleration
22 8: airborne with <4g Acceleration
23 */
24
25 // start getUBX_ACK function +++++++++++++++++++++++++++++++++++++++++++++++++
26 boolean getUBX_ACK(uint8_t *MSG)
27 {
28   uint8_t b;
29   uint8_t ackByteID = 0;
30   uint8_t ackPacket[10];
31   unsigned long startTime = millis();
32
33   // Construct the expected ACK packet
34   ackPacket[0] = 0xB5;  // header
35   ackPacket[1] = 0x62;  // header
36   ackPacket[2] = 0x05;  // class
37   ackPacket[3] = 0x01;  // id
38   ackPacket[4] = 0x02;  // length
39   ackPacket[5] = 0x00;
40   ackPacket[6] = MSG[2];  // ACK class
41   ackPacket[7] = MSG[3];  // ACK id
42   ackPacket[8] = 0;   // CK_A
43   ackPacket[9] = 0;   // CK_B
44
45   // Calculate the checksums
46   for (uint8_t ubxi=2; ubxi<8; ubxi++) {
47     ackPacket[8] = ackPacket[8] + ackPacket[ubxi];
48     ackPacket[9] = ackPacket[9] + ackPacket[8];
49   }
50
51   while (1) {
52
53     // Test for success
54     if (ackByteID > 9) {
55       // All packets in order!
```

```
56        return true;
57      }
58
59      // Timeout if no valid response in 3 seconds
60      if (millis() - startTime > 3000) {
61        return false;
62      }
63
64      // Make sure data is available to read
65      if (Serial.available()) {
66        b = Serial.read();
67
68        // Check that bytes arrive in sequence as per expected ACK packet
69        if (b == ackPacket[ackByteID]) {
70          ackByteID++;
71        }
72        else {
73          ackByteID = 0;  // Reset and look again, invalid order
74        }
75
76      }
77    }
78  }
79  //end getUBX_ACK function
80
81  // start gps_CRC16_checksum function ++++++++++++++++++++++++++++++++++++
82  uint16_t gps_CRC16_checksum (char *string)
83  {
84    size_t i;
85    uint16_t crc;
86    uint8_t c;
87
88    crc = 0xFFFF;
89
90    // Calculate checksum ignoring the first two $s
91    for (i = 5; i < strlen(string); i++)
92    {
93      c = string[i];
94      crc = _crc_xmodem_update (crc, c);
95    }
96
97    return crc;
98  }
99  //end gps_CRC16_checksum function
100
101 // start gps_check_nav function ++++++++++++++++++++++++++++++++++++++++++++
102 uint8_t gps_check_nav(void)
103 {
104   uint8_t request[8] = {
105     0xB5, 0x62, 0x06, 0x24, 0x00, 0x00, 0x2A, 0x84           };
106   sendUBX(request, 8);
107
108   // Get the message back from the GPS
109   gps_get_data();
110
111   // Verify sync and header bytes
112   if( buf[0] != 0xB5 || buf[1] != 0x62 ){
113     GPSerror = 41;
```

```
114   }
115   if( buf[2] != 0x06 || buf[3] != 0x24 ){
116     GPSerror = 42;
117   }
118   // Check 40 bytes of message checksum
119   if( !_gps_verify_checksum(&buf[2], 40) ) {
120     GPSerror = 43;
121   }
122   // Return the navigation mode and let the caller analyse it
123   navmode = buf[8];
124 }
125 //end gps_check_nav function
126 
127 // start _gps_verify_checksum function ++++++++++++++++++++++++++++++++++
128 bool _gps_verify_checksum(uint8_t* data, uint8_t len)
129 {
130   uint8_t a, b;
131   gps_ubx_checksum(data, len, &a, &b);
132   if( a != *(data + len) || b != *(data + len + 1))
133     return false;
134   else
135     return true;
136 }
137 //end _gps_verify_checksum function
138 
139 // start gps_ubx_checksum function ++++++++++++++++++++++++++++++++++++++
140 void gps_ubx_checksum(uint8_t* data, uint8_t len, uint8_t* cka, uint8_t*
      ckb)
141 {
142   *cka = 0;
143   *ckb = 0;
144   for( uint8_t i = 0; i < len; i++ )
145   {
146     *cka += *data;
147     *ckb += *cka;
148     data++;
149   }
150 }
151 //end gps_ubx_checksum function
152 
153 // start checkDynamicModel function ++++++++++++++++++++++++++++++++++++++
154 void checkDynamicModel()
155 {
156   if(alt<=1000&&sats>4) {
157     if(navmode != 3)
158     {
159       setGPS_DynamicModel3();
160       errorstatus |=(1 << 3);  // Set Bit 3 indicating we are in pedestrian
        mode
161     }
162   }
163   else
164   {
165     if(navmode != 6){
166       setGPS_DynamicModel6();
167       errorstatus &= ~(1 << 3); // Unset bit 3 indicating we are in flight
      mode
168 
```

```
169     }
170   }
171 }
172 //end checkDynamicModel function
173
174 // start sendUBX function +++++++++++++++++++++++++++++++++++++++++++++++++
175 void sendUBX(uint8_t *MSG, uint8_t len)
176 {
177   Serial.flush();
178   Serial.write(0xFF);
179   wait(100);
180   for(int i=0; i<len; i++) {
181     Serial.write(MSG[i]);
182   }
183 }
184 //end sendUBX function
185
186 // start setupGPS function +++++++++++++++++++++++++++++++++++++++++++++++++
187 void setupGPS()
188 {
189   // Turning off all GPS NMEA strings apart on the uBlox module
190   // Taken from Project Swift (rather than the old way of sending ascii
      text)
191   int gps_set_sucess=0;
192   uint8_t setNMEAoff[] = {
193     0xB5, 0x62, 0x06, 0x00, 0x14, 0x00, 0x01, 0x00, 0x00, 0x00, 0xD0, 0x08,
      0x00, 0x00, 0x80, 0x25, 0x00, 0x00, 0x07, 0x00, 0x01, 0x00, 0x00, 0x00,
      0x00, 0x00, 0xA0, 0xA9           };
194   sendUBX(setNMEAoff, sizeof(setNMEAoff)/sizeof(uint8_t));
195   while(!gps_set_sucess)
196   {
197     sendUBX(setNMEAoff, sizeof(setNMEAoff)/sizeof(uint8_t));
198     gps_set_sucess=getUBX_ACK(setNMEAoff);
199     if(!gps_set_sucess)
200     {
201       blinkled(2);
202     }
203   }
204   wait(500);
205   setGPS_GNSS();
206   wait(500);
207   setGPS_DynamicModel6();
208   wait(500);
209   setGps_MaxPerformanceMode();
210   wait(500);
211 }
212 // end setupGPS function
213
214 // start resetGPS function +++++++++++++++++++++++++++++++++++++++++++++++++
215 void resetGPS()
216 {
217   uint8_t set_reset[] = {
218     0xB5, 0x62, 0x06, 0x04, 0x04, 0x00, 0xFF, 0x87, 0x00, 0x00, 0x94, 0xF5
      };
219   sendUBX(set_reset, sizeof(set_reset)/sizeof(uint8_t));
220 }
221 // end resetGPS function
222
```

```
223 // start setGPS_DynamicModel6 function ++++++++++++++++++++++++++++++++++++
224 void setGPS_DynamicModel6()
225 {
226   int gps_set_sucess=0;
227   uint8_t setdm6[] = {
228     0xB5, 0x62, 0x06, 0x24, 0x24, 0x00, 0xFF, 0xFF, 0x06,
229     0x03, 0x00, 0x00, 0x00, 0x00, 0x10, 0x27, 0x00, 0x00,
230     0x05, 0x00, 0xFA, 0x00, 0xFA, 0x00, 0x64, 0x00, 0x2C,
231     0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
232     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x16, 0xDC           };
233   while(!gps_set_sucess)
234   {
235     sendUBX(setdm6, sizeof(setdm6)/sizeof(uint8_t));
236     gps_set_sucess=getUBX_ACK(setdm6);
237   }
238 }
239 // end setGPS_DynamicModel6 function
240
241 // start setGPS_GNSS function +++++++++++++++++++++++++++++++++++++++++++++++
242 void setGPS_GNSS()
243 {
244   // Sets CFG-GNSS to disable everything other than GPS GNSS
245   // solution. Failure to do this means GPS power saving
246   // doesn't work. Not needed for MAX7, needed for MAX8's
247   int gps_set_sucess=0;
248   uint8_t setgnss[] = {
249     0xB5, 0x62, 0x06, 0x3E, 0x2C, 0x00, 0x00, 0x00,
250     0x20, 0x05, 0x00, 0x08, 0x10, 0x00, 0x01, 0x00,
251     0x01, 0x01, 0x01, 0x01, 0x03, 0x00, 0x00, 0x00,
252     0x01, 0x01, 0x03, 0x08, 0x10, 0x00, 0x00, 0x00,
253     0x01, 0x01, 0x05, 0x00, 0x03, 0x00, 0x00, 0x00,
254     0x01, 0x01, 0x06, 0x08, 0x0E, 0x00, 0x00, 0x00,
255     0x01, 0x01, 0xFC, 0x11   };
256   while(!gps_set_sucess)
257   {
258     sendUBX(setgnss, sizeof(setgnss)/sizeof(uint8_t));
259     gps_set_sucess=getUBX_ACK(setgnss);
260   }
261 }
262 // end setGPS_GNSS function
263
264 // start setGPS_DynamicModel3 function ++++++++++++++++++++++++++++++++++++
265 void setGPS_DynamicModel3()
266 {
267   int gps_set_sucess=0;
268   uint8_t setdm3[] = {
269     0xB5, 0x62, 0x06, 0x24, 0x24, 0x00, 0xFF, 0xFF, 0x03,
270     0x03, 0x00, 0x00, 0x00, 0x00, 0x10, 0x27, 0x00, 0x00,
271     0x05, 0x00, 0xFA, 0x00, 0xFA, 0x00, 0x64, 0x00, 0x2C,
272     0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
273     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x13, 0x76           };
274   while(!gps_set_sucess)
275   {
276     sendUBX(setdm3, sizeof(setdm3)/sizeof(uint8_t));
277     gps_set_sucess=getUBX_ACK(setdm3);
278   }
279 }
280 // end setGPS_DynamicModel3 function
```

```
281
282 // start setGps_MaxPerformanceMode function ++++++++++++++++++++++++++++++
283 void setGps_MaxPerformanceMode()
284 {
285   //Set GPS for Max Performance Mode
286   uint8_t setMax[] = {
287     0xB5, 0x62, 0x06, 0x11, 0x02, 0x00, 0x08, 0x00, 0x21, 0x91 }; // Setup
      for Max Power Mode
288   sendUBX(setMax, sizeof(setMax)/sizeof(uint8_t));
289 }
290 // end setGps_MaxPerformanceMode function
291
292 // start gps_get_data function ++++++++++++++++++++++++++++++++++++++++++++
293 void gps_get_data()
294 {
295   Serial.flush();
296   // Clear buf[i]
297   for(int i = 0;i<60;i++)
298   {
299     buf[i] = 0; // clearing buffer
300   }
301   int i = 0;
302   unsigned long startTime = millis();
303
304   while ((i<60) && ((millis() - startTime) < 1000) ) {
305     if (Serial.available()) {
306       buf[i] = Serial.read();
307       i++;
308     }
309   }
310 }
311 // end gps_get_data function
312
313 // start gps_check_lock function ++++++++++++++++++++++++++++++++++++++++++
314 void gps_check_lock()
315 {
316   GPSerror = 0;
317   Serial.flush();
318   // Construct the request to the GPS
319   uint8_t request[8] = {
320     0xB5, 0x62, 0x01, 0x06, 0x00, 0x00,
321     0x07, 0x16
                                    };
322   sendUBX(request, 8);
323
324   // Get the message back from the GPS
325   gps_get_data();
326   // Verify the sync and header bits
327   if( buf[0] != 0xB5 || buf[1] != 0x62 ) {
328     GPSerror = 11;
329   }
330   if( buf[2] != 0x01 || buf[3] != 0x06 ) {
331     GPSerror = 12;
332   }
333
334   // Check 60 bytes minus SYNC and CHECKSUM (4 bytes)
335   if( !_gps_verify_checksum(&buf[2], 56) ) {
```

```
336     GPSerror = 13;
337   }
338
339   if(GPSerror == 0){
340     // Return the value if GPSfixOK is set in 'flags'
341     if( buf[17] & 0x01 )
342       lock = buf[16];
343     else
344       lock = 0;
345
346     sats = buf[53];
347   }
348   else {
349     lock = 0;
350   }
351 }
352 // end gps_check_lock function
353
354 // start gps_get_position function ++++++++++++++++++++++++++++++++++++++++
355 void gps_get_position()
356 {
357   GPSerror = 0;
358   Serial.flush();
359   // Request a NAV-POSLLH message from the GPS
360   uint8_t request[8] = {
361     0xB5, 0x62, 0x01, 0x02, 0x00, 0x00, 0x03, 0x0A           };
362   sendUBX(request, 8);
363
364   // Get the message back from the GPS
365   gps_get_data();
366
367   // Verify the sync and header bits
368   if( buf[0] != 0xB5 || buf[1] != 0x62 )
369     GPSerror = 21;
370   if( buf[2] != 0x01 || buf[3] != 0x02 )
371     GPSerror = 22;
372
373   if( !_gps_verify_checksum(&buf[2], 32) ) {
374     GPSerror = 23;
375   }
376
377   if(GPSerror == 0) {
378     if(sats<4)
379     {
380       lat=0;
381       lon=0;
382       alt=0;
383     }
384     else
385     {
386       lon = (int32_t)buf[10] | (int32_t)buf[11] << 8 |
387         (int32_t)buf[12] << 16 | (int32_t)buf[13] << 24;
388       lat = (int32_t)buf[14] | (int32_t)buf[15] << 8 |
389         (int32_t)buf[16] << 16 | (int32_t)buf[17] << 24;
390       alt = (int32_t)buf[22] | (int32_t)buf[23] << 8 |
391         (int32_t)buf[24] << 16 | (int32_t)buf[25] << 24;
392     }
393     // 4 bytes of latitude/longitude (1e-7)
```

```
394     lon_int=abs(lon/10000000);
395     lon_dec=(labs(lon) % 10000000)/10;
396     lat_int=abs(lat/10000000);
397     lat_dec=(labs(lat) % 10000000)/10;
398 
399     // 4 bytes of altitude above MSL (mm)
400     alt /= 1000; // converte to meters
401   }
402 
403 }
404 // end gps_get_position function
405 
406 // start setGPS_PowerSaveMode function +++++++++++++++++++++++++++++++++++
407 void setGPS_PowerSaveMode()
408 {
409   // Power Save Mode
410   uint8_t setPSM[] = {
411     0xB5, 0x62, 0x06, 0x11, 0x02, 0x00, 0x08, 0x01, 0x22, 0x92           };
      // Setup for Power Save Mode (Default Cyclic 1s)
412   sendUBX(setPSM, sizeof(setPSM)/sizeof(uint8_t));
413 }
414 // end setGPS_PowerSaveMode function
415 
416 // start gps_get_time function ++++++++++++++++++++++++++++++++++++++++++++
417 void gps_get_time()
418 {
419   GPSerror = 0;
420   Serial.flush();
421   // Send a NAV-TIMEUTC message to the receiver
422   uint8_t request[8] = {
423     0xB5, 0x62, 0x01, 0x21, 0x00, 0x00, 0x22, 0x67           };
424   sendUBX(request, 8);
425 
426   // Get the message back from the GPS
427   gps_get_data();
428 
429   // Verify the sync and header bits
430   if( buf[0] != 0xB5 || buf[1] != 0x62 )
431     GPSerror = 31;
432   if( buf[2] != 0x01 || buf[3] != 0x21 )
433     GPSerror = 32;
434 
435   if( !_gps_verify_checksum(&buf[2], 24) ) {
436     GPSerror = 33;
437   }
438 
439   if(GPSerror == 0) {
440     if(buf[22] > 23 || buf[23] > 59 || buf[24] > 59)
441     {
442       GPSerror = 34;
443     }
444     else {
445       hour = buf[22];
446       minute = buf[23];
447       second = buf[24];
448     }
449   }
450 }
```

```
451 // end gps_get_time function
452
453 // EOF ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

A.3 Additional Header Files

Listing 6: File *ax25modem.h* containing variable definitions for the AX25 modem.

```
1  /* From Project Swift - High altitude balloon flight software                */
2  /*=========================================================================*/
3  /* Copyright 2010-2012 Philip Heron <phil@sanslogic.co.uk>                  */
4  /*                     Nigel Smart <nigel@projectswift.co.uk>               */
5  /*                                                                          */
6  /* This program is free software: you can redistribute it and/or modify     */
7  /* it under the terms of the GNU General Public License as published by     */
8  /* the Free Software Foundation, either version 3 of the License, or        */
9  /* (at your option) any later version.                                      */
10 /*                                                                          */
11 /* This program is distributed in the hope that it will be useful,          */
12 /* but WITHOUT ANY WARRANTY; without even the implied warranty of           */
13 /* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the            */
14 /* GNU General Public License for more details.                             */
15 /*                                                                          */
16 /* You should have received a copy of the GNU General Public License        */
17 /* along with this program.  If not, see <http://www.gnu.org/licenses/>.    */
18
19 //#define APRS // Uncomment to use APRS.
20
21 #ifndef __AX25MODEM_H
22 #define __AX25MODEM_H
23 #define APRS_TX_INTERVAL 1  // APRS TX Interval in minutes
24 #define APRS_CALLSIGN "OZ7EMA"
25 #define APRS_SSID     (11)
26
27 extern void ax25_init(void);
28 extern void ax25_frame(char *scallsign, char sssid, char *dcallsign, char
       dssid,
29 char *path1, char ttl1, char *path2, char ttl2, char *data, ...);
30 extern char *ax25_base91enc(char *s, uint8_t n, uint32_t v);
31
32 #endif
33
34 // EOF ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Listing 7: File *sine_table.h* containing numeric values of the sinus function.

```
1  /*
2   TITAN HAB flight computer software
3   Modifications copyright 2015 by Emanuel Bombasaro
4   file containg sinus_table
5
6  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
7  created 25-XII-2014 EMBO
8  edited  09-  I-2015 EMBO
9  */
10 0x80,0x81,0x83,0x84,0x86,0x87,0x89,0x8A,0x8C,0x8D,0x8F,0x91,0x92,0x94,0x95
       ,0x97,
```

```
0x98,0x9A,0x9B,0x9D,0x9E,0xA0,0xA1,0xA3,0xA4,0xA6,0xA7,0xA9,0xAA,0xAC,0xAD,0xAF,
0xB0,0xB2,0xB3,0xB4,0xB6,0xB7,0xB9,0xBA,0xBB,0xBD,0xBE,0xBF,0xC1,0xC2,0xC3,0xC5,
0xC6,0xC7,0xC9,0xCA,0xCB,0xCC,0xCE,0xCF,0xD0,0xD1,0xD2,0xD4,0xD5,0xD6,0xD7,0xD8,
0xD9,0xDA,0xDB,0xDD,0xDE,0xDF,0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE6,0xE7,0xE8,
0xE9,0xEA,0xEB,0xEC,0xEC,0xED,0xEE,0xEF,0xF0,0xF0,0xF1,0xF2,0xF2,0xF3,0xF4,0xF4,
0xF5,0xF5,0xF6,0xF7,0xF7,0xF8,0xF8,0xF9,0xF9,0xF9,0xFA,0xFA,0xFB,0xFB,0xFB,0xFC,
0xFC,0xFC,0xFD,0xFD,0xFD,0xFD,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,
0xFF,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFD,0xFD,0xFD,0xFD,0xFC,
0xFC,0xFC,0xFB,0xFB,0xFB,0xFA,0xFA,0xF9,0xF9,0xF9,0xF8,0xF8,0xF7,0xF7,0xF6,0xF5,
0xF5,0xF4,0xF4,0xF3,0xF2,0xF2,0xF1,0xF0,0xF0,0xEF,0xEE,0xED,0xEC,0xEC,0xEB,0xEA,
0xE9,0xE8,0xE7,0xE6,0xE6,0xE5,0xE4,0xE3,0xE2,0xE1,0xE0,0xDF,0xDE,0xDD,0xDB,0xDA,
0xD9,0xD8,0xD7,0xD6,0xD5,0xD4,0xD2,0xD1,0xD0,0xCF,0xCE,0xCC,0xCB,0xCA,0xC9,0xC7,
0xC6,0xC5,0xC3,0xC2,0xC1,0xBF,0xBE,0xBD,0xBB,0xBA,0xB9,0xB7,0xB6,0xB4,0xB3,0xB2,
0xB0,0xAF,0xAD,0xAC,0xAA,0xA9,0xA7,0xA6,0xA4,0xA3,0xA1,0xA0,0x9E,0x9D,0x9B,0x9A,
0x98,0x97,0x95,0x94,0x92,0x91,0x8F,0x8D,0x8C,0x8A,0x89,0x87,0x86,0x84,0x83,0x81,
0x80,0x7E,0x7C,0x7B,0x79,0x78,0x76,0x75,0x73,0x72,0x70,0x6E,0x6D,0x6B,0x6A,0x68,
0x67,0x65,0x64,0x62,0x61,0x5F,0x5E,0x5C,0x5B,0x59,0x58,0x56,0x55,0x53,0x52,0x50,
0x4F,0x4D,0x4C,0x4B,0x49,0x48,0x46,0x45,0x44,0x42,0x41,0x40,0x3E,0x3D,0x3C,0x3A,
0x39,0x38,0x36,0x35,0x34,0x33,0x31,0x30,0x2F,0x2E,0x2D,0x2B,0x2A,0x29,0x28,0x27,
0x26,0x25,0x24,0x22,0x21,0x20,0x1F,0x1E,0x1D,0x1C,0x1B,0x1A,0x19,0x19,0x18,0x17,
0x16,0x15,0x14,0x13,0x13,0x12,0x11,0x10,0x0F,0x0F,0x0E,0x0D,0x0D,0x0C,0x0B,0x0B,
0x0A,0x0A,0x09,0x08,0x08,0x07,0x07,0x06,0x06,0x06,0x05,0x05,0x04,0x04,0x04,0x03,
0x03,0x03,0x02,0x02,0x02,0x02,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x02,0x02,0x02,0x02,0x03,
0x03,0x03,0x04,0x04,0x04,0x05,0x05,0x06,0x06,0x06,0x07,0x07,0x08,0x08,0x09,0x0A,
0x0A,0x0B,0x0B,0x0C,0x0D,0x0D,0x0E,0x0F,0x0F,0x10,0x11,0x12,0x13,0x13,0x14,0x15,
0x16,0x17,0x18,0x19,0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F,0x20,0x21,0x22,0x24,0x25,
0x26,0x27,0x28,0x29,0x2A,0x2B,0x2D,0x2E,0x2F,0x30,0x31,0x33,0x34,0x35,0x36,0x38,
0x39,0x3A,0x3C,0x3D,0x3E,0x40,0x41,0x42,0x44,0x45,0x46,0x48,0x49,0x4B,0x4C,0x4D,
```

```
0x4F,0x50,0x52,0x53,0x55,0x56,0x58,0x59,0x5B,0x5C,0x5E,0x5F,0x61,0x62,0x64,0x65,
0x67,0x68,0x6A,0x6B,0x6D,0x6E,0x70,0x72,0x73,0x75,0x76,0x78,0x79,0x7B,0x7C,0x7E,
// EOF ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

B Flight Control Centre Source Code

Listing 8: File *TitanControlCentre.html* Flight Control Centre html file.

```
<!--
  Main flight control centre file
  The web application allows to vizualize the logged transmission of Fldidi in real time
  created by Emanuel Bombasaro 27.II.2015
  edited 01.III.2015 Emanuel Bombasaro
-->
<!DOCTYPE html>
<html>
  <head>
  <title>Flight Control Centre</title>
    <LINK href="css/mystyle.css" rel="stylesheet" type="text/css">
    <script src="https://maps.googleapis.com/maps/api/js"></script>
    <script src="js/TitanControlCentre.js"></script>
  </head>
  <body>


<div class="header">
  <img src="images/missionlogo1.jpg" alt="Mission 1 logo">
  <h1>Flight Control Centre</h1>
  <div id="file1"> Specify log file: <input type="text" id="fileInput" value="file:///Users/ema/.fldigi/ .log" size="40"></div>
</div>
  <div>
  <pre id="string1">No file selected.<pre>
  </div>

  <div id="upfield"><input id="stastopB" type="button" onclick="startstop();" value="Start" /></div>
<div>
Time: <input type="text" id="time" size="8">
Position (lon,lat): <input type="text" id="pos" size="19">
Altitude: <input type="text" id="alt" size="5"> m,
Satellites: <input type="text" id="sats" size="3">
</div>
<p></p>
<div>
Temperature: <input type="text" id="temp" size="3"> &deg C,
Battery status: <progress id="batteryPro" value="0" max="100"></progress> <input type="text" id="battery" size="5"> mV
</div>
<p></p>
<div id="map-canvas"></div>
<p></p>
<div class="circle gray" id="Bit0"></div><div id="errorc">GPS Error Condition Noted: <font color="blue">Switch to Max Performance Mode</font
```

```
    ></div>
<div class="circle gray" id="Bit1"></div><div id="errorc">GPS Error
    Condition Noted: <font color="red">Cold Boot GPS</font></div>
<div class="circle gray" id="Bit2"></div><div id="errorc">DS18B20 temp
    sensor status: <font color="green">OK</font>, <font color="red">Fault</
    font></div>
<div class="circle gray" id="Bit3"></div><div id="errorc">Current Dynamic
    Model: <font color="green">Flight</font>, <font color="blue">Pedestrian<
    /font></div>
<div class="circle gray" id="Bit4"></div><div id="errorc">PSM status: <font
     color="green">PSM On</font>, <font color="blue">PSM Off</font></div>
<div class="circle gray" id="Bit5"></div><div id="errorc">GPS status: <font
     color="green">Locked</font>, <font color="red">Not Locked</font></div>
<div class="circle gray" id="Bit6"></div><div id="errorc">TSL2561 lumosity
    sensor status: <font color="green">OK</font>, <font color="red">Fault</
    font></div>
<div class="circle gray" id="Bit7"></div><div id="errorc">HTU21DF temp
    humidity sensor status: <font color="green">OK</font>, <font color="red"
    >Fault</font></div>
<div class="circle gray" id="Bit8"></div><div id="errorc">MPL3115A2
    pressure sensor status: <font color="green">OK</font>, <font color="red"
    >Fault</font></div>
<div class="circle gray" id="Bit9"></div><div id="errorc">MCP9808
    temperatur precision sensor status: <font color="green">OK</font>, <font
     color="red">Fault</font></div>
<div class="circle gray" id="Bit10"></div><div id="errorc">BMP085 pressure
    sensor status: <font color="green">OK</font>, <font color="red">Fault</
    font></div>
<div class="circle gray" id="Bit11"></div><div id="errorc">Gyro sensor
    status: <font color="green">OK</font>, <font color="red">Fault</font></
    div>
<div class="circle gray" id="Bit12"></div><div id="errorc">Acc sensor
    status: <font color="green">OK</font>, <font color="red">Fault</font></
    div>
<div class="circle gray" id="Bit13"></div><div id="errorc">Mag sensor
    status: <font color="green">OK</font>, <font color="red">Fault</font></
    div>
<div class="circle gray" id="Bit14"></div><div id="errorc">Logging status:
    <font color="green">OK</font>, <font color="red">Fault</font></div>

<footer>
  <p>&copy by Emanuel Bombasaro 2015</p>
</footer>
  </body>
</html>
```

Listing 9: File *TitanControlCentre.js* Flight Control Centre javascript file.

```
/* java script file for Titan Controll Center
   created by Emanuel Bombasaro 27-II-2015
   edited: 28-II-2015
   edited: 01-III-2015

*/
var fileInput;
var fileDisplayArea;
var file;
var stastopB;
var readString="No file selected.";
```

```
var callsign;
var count;
var time;
var lon;
var lat;
var maxalt;
var sats;
var temperature1;
var battvaverage;
var errorstatus;
var Bitarr=[];
var myTimer, onTimer=0;

var ti, latlon, sat, alts, temp, batt, battPro;
var bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7;
var bit8, bit9, bit10, bit11, bit12, bit13, bit14, bit15;

var mypos, HABpos, markerME, markerHAB, meHAB, meHABpath;
var me, hab, map;

// Check for the various File API support.
if (window.File && window.FileReader && window.FileList && window.Blob) {
  // Great success! All the File APIs are supported.
  } else {
      alert('The File APIs are not fully supported by your browser.');
}

// create google map
google.maps.event.addDomListener(window, 'load', initialize);

window.onload = function() {
  fileInput = document.getElementById('fileInput');
  stastopB = document.getElementById('stastopB');
  fileDisplayArea = document.getElementById('string1');
  ti = document.getElementById("time");
  latlon = document.getElementById("pos");
  sat = document.getElementById("sats");
  alts = document.getElementById("alt");
  temp = document.getElementById("temp");
  batt = document.getElementById("battery");
  battPro = document.getElementById("batteryPro");
  bit0 = document.getElementById("Bit0");
  bit1 = document.getElementById("Bit1");
  bit2 = document.getElementById("Bit2");
  bit3 = document.getElementById("Bit3");
  bit4 = document.getElementById("Bit4");
  bit5 = document.getElementById("Bit5");
  bit6 = document.getElementById("Bit6");
  bit7 = document.getElementById("Bit7");
  bit8 = document.getElementById("Bit8");
  bit9 = document.getElementById("Bit9");
  bit10 = document.getElementById("Bit10");
  bit11 = document.getElementById("Bit11");
  bit12 = document.getElementById("Bit12");
  bit13 = document.getElementById("Bit13");
  bit14 = document.getElementById("Bit14");
  bit15 = document.getElementById("Bit15");

```

```
70    fileDisplayArea.innerText = readString;
71    // set today as log file
72    var today = new Date();
73    fileInput.value=("file:///Users/ema/.fldigi/fldigi" + today.getFullYear()
        + ("0" + (today.getMonth() + 1)).slice(-2) + ("0" + today.getDate()).
       slice(-2) + ".log");
74
75    me = {
76          url: 'images/me.png',
77          origin: new google.maps.Point(0, 0),
78          anchor: new google.maps.Point(30, 45),
79      };
80    hab = {
81          url: 'images/hab.png',
82          origin: new google.maps.Point(0, 0),
83          anchor: new google.maps.Point(20, 45),
84      };
85  }
86
87  // start stop button click
88  function startstop() {
89    // automatic update
90    if (onTimer==0) {
91      onTimer=1;
92      myTimer = setInterval(function () {updateAll()}, 500);
93      stastopB.value="Stop";
94    } else {
95      onTimer=0;
96      clearInterval(myTimer);
97      stastopB.value="Start";
98    }
99  }
100
101 // update function
102 function updateAll() {
103   file = fileInput.value;
104   var rawFile = new XMLHttpRequest();
105     rawFile.open("GET", file, false);
106     rawFile.onreadystatechange = function ()
107     {
108         if(rawFile.readyState === 4)
109         {
110             if(rawFile.status === 200 || rawFile.status == 0)
111             {
112                 readString = rawFile.responseText;
113                 analyseString();
114         updateFields();
115             }
116         }
117     }
118     rawFile.send(null);
119 }
120
121 // function updating fields
122 function updateFields()
123 {
124   fileDisplayArea.innerText = readString;
125   ti.value=time;
```

```
126   latlon.value=(lat + "," + lon);
127   sat.value=sats;
128   alts.value=maxalt;
129   temp.value=temperature1;
130   batt.value=battvaverage;
131   battPro.value=battvaverage/3600*100;
132   getBits();
133   if (Bitarr[0]==0) {bit0.className = "circle green";} else {bit0.className
        = "circle blue";}
134   if (Bitarr[1]==0) {bit1.className = "circle green";} else {bit1.className
        = "circle red";}
135   if (Bitarr[2]==0) {bit2.className = "circle green";} else {bit2.className
        = "circle red";}
136   if (Bitarr[3]==0) {bit3.className = "circle green";} else {bit3.className
        = "circle blue";}
137   if (Bitarr[4]==0) {bit4.className = "circle green";} else {bit4.className
        = "circle blue";}
138   if (Bitarr[5]==0) {bit5.className = "circle green";} else {bit5.className
        = "circle red";}
139   if (Bitarr[6]==0) {bit6.className = "circle green";} else {bit6.className
        = "circle red";}
140   if (Bitarr[7]==0) {bit7.className = "circle green";} else {bit7.className
        = "circle red";}
141   if (Bitarr[8]==0) {bit8.className = "circle green";} else {bit8.className
        = "circle red";}
142   if (Bitarr[9]==0) {bit9.className = "circle green";} else {bit9.className
        = "circle red";}
143   if (Bitarr[10]==0) {bit10.className = "circle green";} else {bit10.
       className = "circle red";}
144   if (Bitarr[11]==0) {bit11.className = "circle green";} else {bit11.
       className = "circle red";}
145   if (Bitarr[12]==0) {bit12.className = "circle green";} else {bit12.
       className = "circle red";}
146   if (Bitarr[13]==0) {bit13.className = "circle green";} else {bit13.
       className = "circle red";}
147   if (Bitarr[14]==0) {bit14.className = "circle green";} else {bit14.
       className = "circle red";}
148   if (Bitarr[15]==0) {bit15.className = "circle green";} else {bit15.
       className = "circle red";}
149 }
150
151 // analyse string and set variables
152 function analyseString()
153 {
154   fileLines=readString.split("\n");
155   // we actually read the penultimum line
156   var i=1;
157   while (fileLines[fileLines.length-i]=="") {i=i+1};
158   readString=fileLines[fileLines.length-i];
159   LineEle=readString.split(",");
160   if (LineEle.length==10) {
161     callsign=LineEle[0];
162     count=LineEle[1];
163     time=LineEle[2];
164     lat=LineEle[3];
165     lon=LineEle[4];
166     maxalt=LineEle[5];
167     sats=LineEle[6];
```

```
168     temperature1=LineEle[7];
169     battvaverage=LineEle[8];
170     errorstatus=LineEle[9].substr(0,LineEle[9].indexOf("*"));
171     // set markers
172     HABpos = new google.maps.LatLng(lat,lon);
173     markerHAB.setPosition(HABpos);
174     getLocation();
175     markerME.setPosition(mypos);
176         meHABpath = [mypos,HABpos];
177     meHAB.setPath(meHABpath);
178     map.setCenter(HABpos);
179     readString = (readString.substring(0,40) + "\n" + readString.substring
      (41,readString.length) + "\nValues updated!");
180   } else {
181     //alert('Set not complied old values conserved!');
182     readString = (readString.substring(0,40) + "\n" + readString.substring
      (41,readString.length) + "\nValues NOT updated!");
183   }
184 }
185
186 //bit shifter
187 function getBits()
188 {
189   for (var i = 0; i < 16; ++i) {
190       Bitarr[i] = (errorstatus >> i) & 1;
191     }
192 }
193
194 // get current location
195 function getLocation() {
196     if (navigator.geolocation) {
197         navigator.geolocation.getCurrentPosition(setPosition);
198     } else {
199          alert("Geolocation is not supported by this browser.");
200     }
201 }
202
203 // assign current location to variable
204 function setPosition(position) {
205   mypos = new google.maps.LatLng(position.coords.latitude,position.coords.
      longitude);
206 }
207
208 // define goolge map
209 function initialize()
210 {
211   var mapCanvas = document.getElementById('map-canvas');
212   var mapOptions = {zoom: 12,
213     mapTypeId: google.maps.MapTypeId.ROADMAP
214     }
215   map = new google.maps.Map(mapCanvas, mapOptions)
216
217   // Try HTML5 geolocation
218   if(navigator.geolocation) {
219       navigator.geolocation.getCurrentPosition(function(position) {
220         mypos = new google.maps.LatLng(position.coords.latitude,position.
      coords.longitude);
221     HABpos = new google.maps.LatLng(position.coords.latitude,position.
```

```
    coords.longitude);

    markerME = new google.maps.Marker({
      position: mypos,
          map: map,
          icon: me,
          title: 'You are here!'
    });
    // where is HAB
    markerHAB = new google.maps.Marker({
        position: mypos,
        map: map,
        icon: hab,
        title: 'Titan is here!'
      });
      // line connecting both
      meHABpath = [mypos,HABpos];
      meHAB = new google.maps.Polyline({
        path: meHABpath,
        geodesic: true,
        strokeColor: '#FF0000',
        strokeOpacity: 1.0,
        strokeWeight: 2
      });

      meHAB.setMap(map)

    map.setCenter(HABpos);
      }, function() {
          handleNoGeolocation(true);
      });
    } else {
      // Browser doesn't support Geolocation
      handleNoGeolocation(false);
  }

  function handleNoGeolocation(errorFlag) {
    if (errorFlag) {
        var content = 'Error: The Geolocation service failed.';
      } else {
        var content = 'Error: Your browser doesn\'t support geolocation.';
      }

    var options = {
        map: map,
        position: new google.maps.LatLng(0.0,0.0),
        content: content
      };

      var infowindow = new google.maps.InfoWindow(options);
      map.setCenter(options.position);
  }

}

// EOF
```

Listing 10: File *mystyle.css* Flight Control Centre css file.

```
/*
 sylte file for Flight Control Centre
 cerated by Emanuel Bombasaro 28-II-2015
 edited 01-III-2015
*/

html {
  font-family: Helvetica, sans-serif;
  font-size: 100%;
}

body{
width:800px;
}

.header img {
  float: left;
  width: 150px;
  height: 150px;
}

.header h1 {
  position: relative;
  top: 5px;
  left: 10px;
}

#file1 {
  position: relative;
  top: -5px;
  left: 10px;
  height: 40px;
}

#upfield {
  position: relative;
  top: -30px;
  left: 160px;
}

#string1 {
  position: relative;
  top: -25px;
  left: 15px;
  height: 40px;
}

#errorc {
  position: relative;
  top: -14px;
  left: 20px;
}

#map-canvas {
        width: 800px;
        height: 600px;
}
```

```
58
59 input {
60     text-align:right;
61 }
62
63 .circle {
64     width: 12px;
65     height: 12px;
66     border-radius: 50%;
67 }
68
69 .red {
70   background-color: red;
71 }
72 .green {
73   background-color: green;
74 }
75 .blue {
76   background-color: blue;
77 }
78 .gray {
79   background-color: gray;
80 }
```