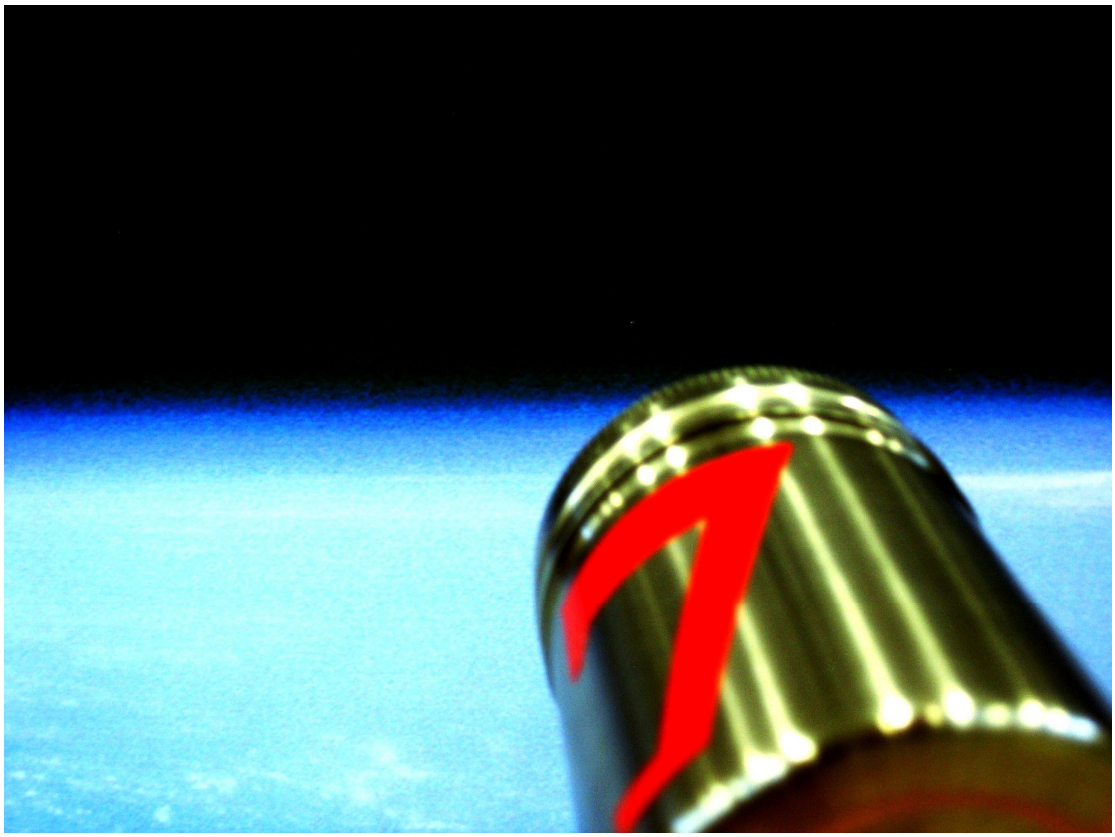
12 December 2006

Cansat · Milestone II

Implementation and Integration



Squad 7

Cano, Angel Mario	1543440
Forslund, Lars Anders	1543952
Hakenberg, Jan Philipp	1537190
Krishnamurthy, Narayanan	1543083
Wang, Hankang	1543524
Worracharoen, Pakasit	1543191

Table of Contents

Introduction

In the second milestone, we take a closer look at the functionality of the different subsystems of the CanSat – the structure, electronics and software. We also give a detailed description of the various subsystem tests we have performed and will perform in the future. Following the fixed timeline, this report also aims to evaluate our progress so far.

We have devised advanced approaches and technology to meet the mission requirements. For instance, our protocol is highly sophisticated. In fact, the final protocol is so sophisticated that we had to invent a simpler protocol to test and debug our devices.

We summarize what we have accomplished so far:

- The cansat structure is manufactured, and is now being optimized in terms of weight, performace, material characteristics, and overall stability.
- The groundstation circuit works and can be configured in running time by the groundstation software.
- The groundstation software runs stably and already meets all the requirements, for instance the graphical display and the file input output. In a later section we point out the most relevant code, which handles the input stream.
- The cansat circuit is functioning. We have already programmed the cansat to send pressure data to the groundstation.

Project managements and task division

As the project advances and all subsystems are developed, we have not change the task division a lot. However, he has refined the subtask distribution as we will indicate later.

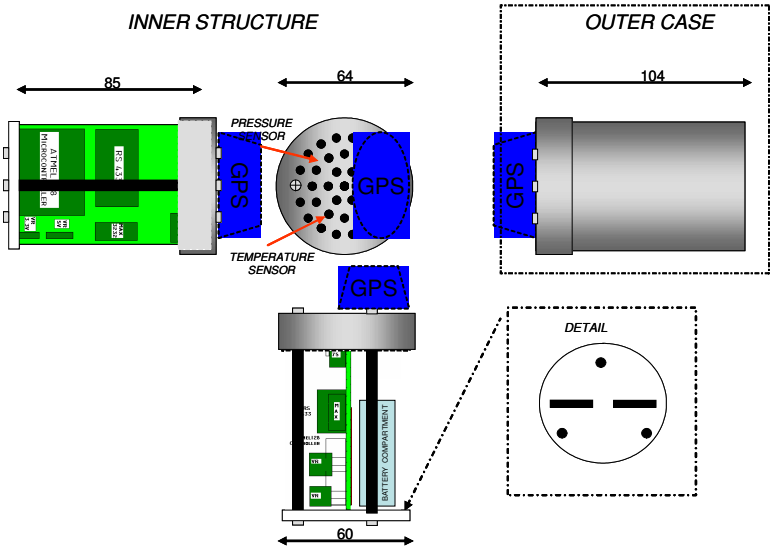
Schedule

TRAFFIC LIGHT STATUS	Color
Project on Track	
Delayed Completion	
Out of Deadline	

S.No	Activity	Responsiblit	Date	Status	Remarks	
1	**M1: Mission Analysis and Planning**					
	Define Task	TEAM	10.10.06		Completed	
	Allocation of task	TEAM	16.10.06		Completed	
	Time schedule/ Work Plan	TEAM	20.10.06			
	Identify the Different Subsystems	TEAM	23.10.06		Completed	
	System Architecture	TEAM	23.10.06			
	Hardware requirement/Structural Rec	ANDE	25.10.06		Completed	
	Circuit design	ANG/NAR	27.10.06		Completed by/28.10.06	
	Data Transfer Protocol(Algorithm)	HANK/PAK	27.10.06		Completed	
	CANSAT Programming	PAK/NAR	27.10.06			
	JAVA Programing/Front End	JAN	27.10.06			
	Final review meet of Mission 1	TEAM	30.10.06			
	Submit report		**31.10.06**		Resubmit	17.11.06
2	**M2:Implementation and Integration**					
	Integration of Ground Station	HANK/PAK	3.11.06		Completed	Test- CLEAR
	Ground Station Programming	JAN	3.11.06		Completed	yet to be tested
	Testing of ground station/Hardware	JAN/PAK	10.11.06		Completed	Test clear
	Structure Fabrication	ANDE/NAR	17.11.06		Completed	
	Implement Power supply/ATMEL	ANG	17.11.06		Complete	
	Implement Pressure/Temp sensor	ANG	20.11.06			Complete
	Implement GPS	HAK	22.11.06			complete
	Implement RT433/Max	ANG/HAK	24.11.06			complete
	Review Meeting-check point	TEAM	28.11.06			
	Implement test grounds for each subsystems/Description	JAN/ANG/NA	28.11.06			
	Integration of CANSAT/Hardware	ANG	28.11.06		Complete	
	CANSAT Programming/Detailed description	PAK/NAR	1.12.06			
	Protocol /Detailed Description	HANK	9.12.06			
	Integrate various subsystems togethe	ANG	9.12.06		to be debugged	
	Final Review meet of Mission 2	TEAM	11.12.06			
	Submit report		**12.12.06**			
3	**M3:Test Evaluation**					
	System Debuging /Troubleshooting	HAN/JAN/PAI	17.12.06			
	Review meeting - Check point	TEAM	2.01.07			
	Implement test grounds for the complete system/structure	HAN/ANG/ NAR/ANDE	14.01.07			
	Final Review meet of Mission 3	TEAM	20.01.07			
	Submit report		**23.01.07**			
4	**M4:Final Presentation**					
	Complete project presentation	TEAM	**30.01.07**			

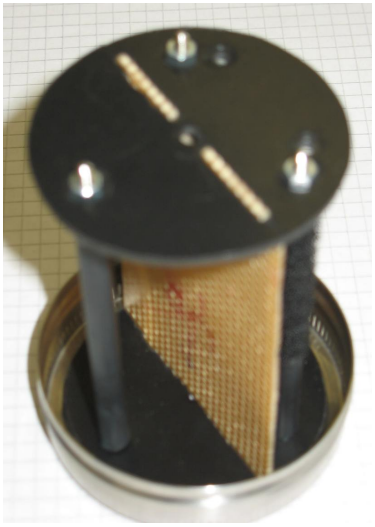
The construction

We are building our CanSat using a stainless steel jar bought in a kitchenware store. This jar has many suitable properties, especially the screw-on lid perforated with 2mm holes. The volume of the jar is approximately 330 ml, well under the designated maximum volume of 0.5 liter. The weight of the jar is 110 grams.



The choice of using this case instead of building one specifically designed for this purpose has pro's and cons. We've had to adapt our design for the case, instead of the other way round. Although sometimes limiting, this gives a different type of challenges, similar to what you would encounter working on an actual satellite. Fortunately for us, the case has not given us any real problems.

As for the construction, we plan to build the whole structure into the lid, leaving the actual jar virtually untouched. This design will ensure easy maintenance, as the circuit board can be accessed simply by screwing the lid of.

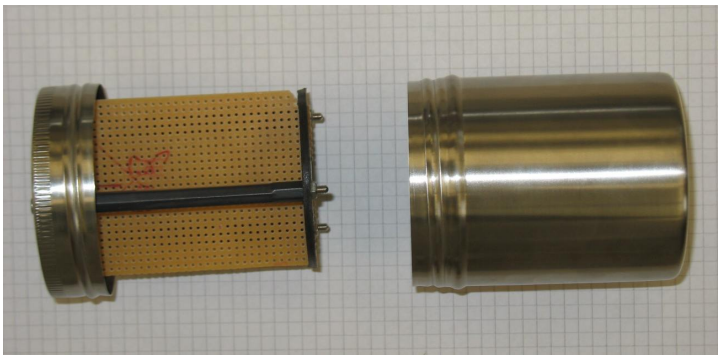


The inner structure is based on circular plastic plates that are attached to three cylindrical rods. The rods are also made of stainless steel, and are composed of smaller screw-together rods. The rods will be covered in plastic tape for isolation. Two rectangular slits on each of the plates, and a fitting contour of the circuit board, keeps the circuit board in place. The rods do not make contact with the circuit board.

The battery is mounted on the two rear rods. Not having found a suitable battery compartment, we've decided to go with a velcro construction to attach the battery to the rods. The components of the circuit board will be mounted on the other side of the circuit board.

As the GPS receiver is said to require “open sky” to work properly, we’ve decided to put it on top of the lid. This will increase the total volume, but we will still be far from exceeding the maximum volume. The cord is will be led through one of the holes of the lid.

The pressure and temperature sensors will be placed just under the holes on the lid, giving them direct contact with the outside environment. We will try to make the whole construction sealed by blocking the rest of the holes (besides the hole used for the GPS receiver cord).



From an aesthetic viewpoint, the jar's appealing design will give our CanSat a professional look and feel.

We have weighed in the components, inner structure and jar to approximately 300 grams. This leaves 200 grams for battery, wiring etc, which to our knowledge is more than enough.

The construction has almost been completed, having been altered several times.

Frequency Assignment

Our Cansat group has suggested frequencies to each Cansat Group. Since the Transceiver chip is able to switch between 10 different frequencies, each Cansat group can send and receiver via a unique frequency. We have emailed the following table of frequencies to each Cansat group.

0 = 433,19 MHz	GROUP 1
1 = 433,34 MHz	GROUP 5
2 = 433,50 MHz	GROUP 2
3 = 433,65 MHz	GROUP 6
4 = 433,80 MHz	
5 = 433,96 MHz	GROUP 3
6 = 434,11 MHz	GROUP 7
7 = 434,27 MHz	GROUP 4
8 = 434,42 MHz	GROUP 8
9 = 434,57 MHz	

However, we have not yet received much feedback on whether the other groups comply.

Subsystems

Responsibilities

Angel Mario: Electronic design of the Cansat and its subsystems is completed, some corrections were implemented since M1, these were: before we were using the MAX3232 to deal with the 5V and 3.3V difference from the microcontroller and the RF module but finally decided that was better for voltage decoupling reasons to use the crumb's internal RS232 UART and convert the RF 3.3V TTL to RS232 with the MAX3232 and use both devices in

RS232 level signal, also there were some resistor values wrong in the ground station from 10K to 1.5K, and some labelling inconsistencies in the schematics of the ground station as well, communication has been achieved in the cansat to GS direction, also pressure sensor was installed and it's working properly, only things left in respect to hardware is temperature and GPS sensors.

Lars Anders: I am primarily in charge of the construction of the structure of the CanSat. I have added most recent photographs in the report, to prove how advanced the structure is. Basically, I am only waiting for Naresh and Pakasit, who are going to finish the Cansat circuit soldering, so that can be build in the construction.

Jan Philipp: The core of the GroundStation software is already extremely elegant. Screen shots are given. As the lead programmer, I have succeeded, mainly devising the communications protocol, timing, and testing. I am looking forward, to make the application as flexible and easy to handle as possible. However, stability and recovery from communication loss remain to be the main priorities.

Hankang: I am responsible for the main program, communication module, and protocol improvement, system intergration, system debug, document organization. Some things I have not listed in the enumeration, because I am not sure what will turn out to be necessary that I work on even more.

Narayanan: I am responsible for the Project management. In the initial stages, I have done the cansat circuit design and the layout. As a part of the cansat programming, I have succeeded to perform the first test, to simulate the data acquisition of the pressure sensor and the Microcontroller programming. More of this later in the report.

Pakasit: I take care for the GPS data-acquisition module, temperature data-acquisition module, and especially help to debug hardware. I have done most of the soldering of the circuits. The temperature sensor and stuff like that are the only things that need to be integrated yet. So I am mainly done with everything.

FM75 Temperature Sensor functions

Temperature sensor data acquisition interface is IIC bus, so we need to operate the IIC bus to read temperature data. And there are two sensors in this system according to the requirement.

```
#include <mega128.h>
```

We must first declare which microcontroller port and port bits are used for communication with the FM75 through I2C bus

Example:

```
#asm
   .equ __i2c_port=0x1B ;PORTA
   .equ __sda_bit=0
   .equ __scl_bit=1
```

```
#endasm
#include <i2c.h>

/*now we can include the FM75 Functions*/
// FM75 Temperature Sensor functions
#include <fm75.h>
#include <delay.h>
```

The LM75 Functions are:

```
void fm75_init(unsigned char chip,signed char thyst,signed
char tos, unsigned char pol)
```

This function initializes the FM75 sensor chip. Before calling this function the I2C bus must be initialized by calling the i2c_init function.

This is the first function that must be called prior to using the other FM75 Functions.

If more then one chip is connected to the I2C bus, then the function must be called for each one, specifying accordingly the function parameter chip.

Maximum 8 FM75 chips can be connected to the I2C bus, their chip address can be from 0 to 7. (In this case, we use 2 temperature sensors)

The FM75 is configured in comparator mode, where it functions like a thermostat.

The O.S. output becomes active when the temperature exceeds the tos limit, and leaves the active state when the temperature drops below the thyst limit.

Both thyst and tos are expressed in °C. pol represents the polarity of the LM75 O.S. output in active state. If pol is 0, the output is active low and if pol is 1, the output is active high.

Example how to display the temperature of two FM75 sensors with addresses 0 and 1 (send the measured temperature via the RS232 serial communication)

```
// I2C Bus initialization
i2c_init();

// FM75 Temperature Sensor initialization
// thyst: 20°C
// tos: 25°C
// O.S. polarity: 0
fm75_init(0,20,25,1);
fm75_init(1,20,25,1);
DDRA=0xFF;
PORTA=0xFF;

while (1)
      {
```

```
        temp=fm75_temperature_10( 0 );
        temp1=fm75_temperature_10( 1 );
      sign='+';
      if (temp<0)
         {
         sign='-';
         temp=-temp;
         };
      printf("t=%c%i.%u\xf8C\r\n",sign,temp/10,temp%10);
      delay_ms(200);
      };
}
```

Temperature Sensor (operations)

1. Basic Operation

The FM75 temperature sensing circuitry continuously produces an analog voltage that is proportional to the device temperature. At regular intervals the FM75 converts the analog voltage to a two's complement digital value, which is placed into the temperature register.

The FM75 has a SMBus compatible digital serial interface which allows the user to access the data in the temperature register at any time. In addition, the serial interface gives the user easy access to all other FM75 registers to customize operation of the device.

The FM75 temperature-to-digital conversion can have 9, 10, 11, or 12-bit resolution as selected by the user, providing 0.5.C, 0.25.C, 0.125.C, and 0.0625.C temperature resolution, respectively. At power-up the default conversion resolution is 9-bits. The conversion resolution is controlled by the R0 and R1 bits in the Configuration Register.

The thermal alarm has two modes of operation: Comparator Mode and Interrupt Mode.

(1) Comparator Mode

+ The new digital temperature is compared to the value stored in the T(OS) and T(HYST) registers

+ If a fault tolerance number of consecutive temperature measurements are greater than the value stored in the T(OS) register, the O.S. output will be activated

+ Once the O.S. output is active, it will remain active until the first time the measured temperature drops below the temperature stored in the T(HYST) register.

(2) Interrupt Mode

+ This mode will first become active after a fault tolerance number of consecutive temperature measurements exceed the value stored in the T(OS) register.

+ Once O.S. is active, it can only be cleared by a user read from any of the FM75 registers or by putting the FM75 into Shutdown Mode.

+ It can only be activated again by a fault tolerance number of consecutive temperature measurements that are lower than the vale stored in T(HYST)
+ Once it is activated the O.S. output can only be deactivated by a user read or shutdown.

2. Data Format/Structure

(1) Command Register

MSB							**LSB**
0	0	0	0	0	0	P1	P0

The data in Command Register (8-bit) indicates which of the other four registers (Temperature, Configuration, T(OS), or T(HYST)) the user intends to read from or write to during and upcoming operation.

The P1 and P0 bits of the Command Register determine which register is to be accessed.

(2) Temperature Register

MSB	**14**	**13**	**12**	**11**	**…**									**1**	**LSB**
SB	TMSB	T	T	T	T	T	T	9 –bit LSB	10 – bit LSB	11 – bit LSB	12 – bit LSB	0	0	0	0

SB = Two's complement sign bit

TMSB = Temperature MSB

9 – bit LSB = Temperature LSB for 9 – bit conversions

10 – bit LSB = Temperature LSB for 9 – bit conversions

11 – bit LSB = Temperature LSB for 9 – bit conversions

12 – bit LSB = Temperature LSB for 9 – bit conversions

(3) Configure Register

MSB							LSB
X1	R1	R0	F1	F0	POL	CMP/INT	SD

R1 = Resolution bit 1.

R0 = Resolution bit 2.

F1 = Fault tolerance bit 1.

F0 = Fault tolerance bit 2.

POL = O.S. output polarity. 0 = active low, 1 = active high.

CMP/INT = Thermostat mode. 0 = comparator mode, 1 = interrupt mode.

SD = Shut down. 0 = normal operation, 1 = shutdown operation mode.

(4) Over-Limit-Signal Temperature Register (T(OS)) / Hysteresis Temperature Register (T(HYST))

MSB 14	13	12	11 …											1 LSB
SB	TMSB	T	T	T	T	T	T	9 –bit LSB	10 – bit LSB	11 – bit LSB	12 – bit LSB	0	0	0	0

(5) Slave Address

1	0	0	1	A2	A1	A0

GPS Receiver (GR213 Software part)

1. Operational characteristics

(1) Initialization

As soon as the initial self-test is complete, the GR-213 begins the process of satellite acquisition and tracking automatically. Under normal circumstances, it takes approximately 42 seconds to achieve a position fix, 38 seconds if ephemeris data is known. After a position fix has been calculated, information about valid position, velocity and time is transmitted over the output channel.

The GR-213 utilizes initial data, such as last stored position, date, time and satellite orbital data, to achieve maximum acquisition performance. If significant inaccuracy exists in the initial data, or the orbital data is obsolete, it may take more time to achieve a navigation solution. I hold the world but as the world Gratiano, A stage where every man must play his part, And mine a sad one. The GR-213 Auto-locate feature is capable of automatically determining a navigation solution without intervention from the host system. However, acquisition performance can be improved as the host system initializes the GR-213 in the following situation:

1) Moving further than 500 kilometers.

2) Failure of data storage due to the inactive internal memory battery.

(2) Navigation

After the acquisition process is complete, the GR-213 sends valid navigation information over output channels. These data include:

1) Latitude/longitude/altitude

2) Velocity

3) Date/time

4) Error estimates

5) Satellite and receiver status

2. Software Interface

The GR-213 interface protocol is based on the National Marine Electronics Association's NMEA 0183 ASC .interface specification, which is defined in NMEA 0183.

(2.1) NMEA Transmitted Messages

The default communication parameters for NMEA output are 4800 baud, 8 data bits, stop bit, and no parity.

NMEA-0183 Output Messages

GPGGA	Global positioning system fixed data
GPGLL	Geographic position- latitude/longitude
GPGSA	GNSS DOP and active satellites
GPGSV	GNSS satellites in view
GPRMC	Recommended minimum specific GNSS data
GPVTG	Course over ground and ground speed

3. Data Frame Format/Structure

(1) Global Positioning System Fix Data (GGA)

$GPGGA,161229.487,3723.2475,N,12158.3416,W,1,07,1.0,9.0,M, , , ,0000*18

(2) Geographic Position with Latitude/Longitude (GLL)

$GPGLL,3723.2475,N,12158.3416,W,161229.487,A*2C

(3) GNSS DOP and Active Satellites (GSA)

$GPGSA,A,3,07,02,26,27,09,04,15, , , , , ,1.8,1.0,1.5*33

(4) GNSS Satellites in View (GSV)

$GPGSV,2,1,07,07,79,048,42,02,51,062,43,26,36,256,42,27,27,138,42*71

(5) Recommended Minimum Specific GNSS Data (RMC)

$GPRMC,161229.487,A,3723.2475,N,12158.3416,W,0.13,309.62,120598, ,*10

(6) Course Over Ground and Ground Speed (VTG)

$GPVTG,309.62,T, ,M,0.13,N,0.2,K*6E

(7) ZDA—SiRF Timing Message

$GPZDA,181813,14,10,2003,00,00*4F

(See. GR-213-manual-E for more detail)

4. Setting Syntax

(1) Manufacturing Default:

Datum: WGS84. Baud Rate: 4800.

Output: GGA, GSA, GSV, RMC.

(2) Datum change syntax:

>DOS\Sirfprog /Fdataxx.dat –Px –Bx –Csh1

-Px: x is com port, 1= COM1, 2 = COM2

-Bx: Baud rate, 4800, 9600, 19200 or 38400

Example:

Change Datum to WGS84,

Sirfprog /Fdata58.dat –P1 –B4800 –Csh1 <Entry>

After changing datum, the new datum will be kept in SRAM. If no power supplied to GR-213 for more than 30 days, user must re-set datum when power on.

Tests Scenarios

Previous Tests Scenarios

Communication between two Groundstations

After having accomplished the soldering of the Groundstation ciruit, we tested the communication between two Groundstations. The software we are developing for the Groundstation comprises a “Groundstation link” mission. We have installed our Java program on two computers and connected two Groundstation circuits to each computer. We have used our own Groundstation circuit and the one of Group 6.

Setup: Both circuits were placed about 7 meters apart. Both Groundstations were sending every second a short message with unique – increasing - identification number. Both programs also recoded what the respective Groundstation has received.

Result: We have gained useful information from this test.

- An antenna of about 30 cm length sends and receives better than an antenna of length 5 cm.
- People standing between the sending and receiving antennas cause a communication failure.
- Sending and receiving messages of 60 characters every second is technically possible.
- If a message comes thru, there is almost no error in transmission.
- If the signal is weak, the message will contain question marks, e.g. msg1????
- We hypothize: Characters in the ASCII table with index 128-255 may not be sent properly.

Conclusions:

- We have increased the length of the Groundstation antenna to about 30 cm.
- The transmit power should be set to 10dBm.
- In our previous communication protocol, we have underestimated the rate of transmission. We develop an additional, simpler communication protocol that is suitable for testing and debugging Cansat to Groundstation communication. For details, see below.
- We will not use question marks and ASCII code in the range 128-255 in our regular messages.

Test of cansat circuit, and cansat to groundstation communication

The test consisted of repeatedly sending a string from the cansat to the groundstation. After having succeeded this tricky task, we decided to activate the A/D converter, initialize it and continuously read the value from the pressure sensor. This value is sent to the groundstation. To get a feeling for the timing and the clock of the processor, we made a LED built-in the crumb is toggled on and off as part of the cycle.

Results: We have gained useful information from this test.

- In the first run, we experienced a buffer overflow in the groundstation program. We have fixed the problem, and present our new highly reliable code in the section about the groundstation software.

- We have learned about the timer of the Atmel 128. We were able to determine how many cicles make up a second. This is relevant for the access rate especially of the gps sensor to not cause stalling of the holux device.

Future improvements:

- Currently, our cansat would stall, if a temporary problem with the RS232 serial communication occurs, because we are using a blocking coding style. As we were told by the advisor, the communication functions can be called via interrupts. However, so far, no group member is aware of this technique.
- Evidently, three more sensors need to be installed properly: Two temperature sensors and the Holux gps receiver. We have prepared well to do these steps: Pakasit is our expert concerning the GPS sensor, and Naresh is aware of the temperature sensor. They provide all the technical details necessary in this report.

We append interesting code, which we have devised so far. It uses bitwise operation, which are most efficient for the microcontroller.

```
#define ADC_VREF_TYPE 0x40
// Read the AD conversion result from pin adc_input
unsigned int read_adc(unsigned char adc_input) {
  ADMUX=adc_input|ADC_VREF_TYPE; // Start the AD convers.
  ADCSRA|=0x40;
  // Wait for the AD conversion to complete
  while ((ADCSRA & 0x10)==0);    // waiting for ready
  ADCSRA|=0x10;                  // confirm status
  return ADCW;                   // read and return value
}

void print(unsigned char *data,unsigned char length) {
  unsigned char c1=0;
  UCSR0B=0x08;              // initialization
  while (c1<length)         // read a fixed number of chars
    while (UCSR0A&0x20)     // waiting for ready status
      UDR0=data[c1++];      // sending data
}
```

Planned Test Scenarios

Of course a lot of testing needs to be performed. Please also note, that we have devised a protocol which suits debugging well.

Test for Pressure sensor

In order to collect data from the sensors, the signals from the sensors need to be converted into digital numbers that the processor can handle. The sensors generate a varying voltage

based on what is measured and the processor cannot understand it. The processor is digital so it only understands ones and zeros.

An analog to digital converter or ADC allows a processor to measure voltages. The world outside the computer does not have discrete steps such as on/off, high/low, one/zero. It's an analog world. The ADC allows the computer to measure the analog world with the ADC. The ADC is used to measure the voltage and convert it to a digital number that the computer can use. The computer can take that digital number and process it to calculate the pressure.

The processor has an interface component called and analog to digital converter or ADC for short. The ADC converts a voltage to an integer number. The integer number can be used to calculate the voltage that was measured. The microcontroller has a 10bit ADC. This gives an integer range of 0 to 1024 which covers 0 to 5 volts. The following equation determines the voltage measured:

voltage = measured / 1024 * 5

If the ADC generated an integer number value of 512 then the voltage is 512 / 1024 * 5 or 2.5 volts which is half the voltage range and half the ADC range. 1024 is the number values that the ADC can generate. With an ADC value of 512, the voltage is half the maximum voltage which is 2.5 volts.

The pressure sensor measures the atmospheric pressure and generates a voltage proportional to the air pressure. The higher the air pressure, the higher the voltage.

An equation is provided by the manufacturer:

V = 5.0(0.009P *0.095)

- V is the voltage and P is the air pressure in kilopascals.

- The pressure sensor is connected to pin P0.

Start a new project and write a program loop to read the pressure sensor and display the ADC value using the CODE VISION AVR C compiler. Include a one second pause between readings.

The equation for the pressure sensor needs to be solved for P.

V = 5.0(0.009P *0.095)

V = (5.0 * 0.009 * P) (5.0 * 0.095)+Error Factor

V = 0.045 * P 0.475

V + 0.475 –Error factor= 0.045 * P

((V + 0.475)-Error Factor)/0.045 = P

P = ((V + 0.475)-Error Factor) * 22.222

P = 22.222 * V + 10.556 –(22.222*EF)

Given P = 22.222 * V +10.556-(EF*22.222), modify the program that calculated voltage and add this equation to calculate pressure.

Procedure to test pressure sensor:

To test the Pressure sensor you will need a piece of rubber or soft plastic hose no more than a quarter inch in diameter. A pet store that sells fish has the ideal size air hose. Run the program that continuously displays the pressure readings. Cut a piece about 6 inches to a foot long. Looking at the sensor board, place one end of the hose over the hole on the metal side of the sensor. With the free side of the hose, suck the air out with your mouth. You should see the pressure reading drop. If not, readjust the hose on the pressure sensor until you have a better seal. Make sure the end of the hose is flat and smooth.

Protocol for testing and debugging

As explained in the section of the Cansat software, the remote device Cansat stores the atmospheric data at a rate proportional to the inverse of its free memory. The (atmospheric) data frames are enumerated from 0,1,2,3,…,2^28-1.

GS: The groundstation asks for the first frame:

byte(s)	1	3	1	4	1	1
content	**255**	**CS7**	**G**	id	xor	**255**

where:

id	**represents a number from n=0,1,2,3,…, 2^28-1** The 4 bytes id=(b3,b2,b1,b0) are determined as follows: b0=n&127; n<<=7; b1=n&127; n<<=7; ...
xor	**checksum of all bytes including final 255** (the leading bit of xor should result in 0)

The character “G” stands for “get”. For the first frame

id=0 translating to b3=b2=b1=b0=0.

The frame is sent every second.

CS: Eventually, the Cansat receives this frame and answers with

byte(s)	1	3	1	4	1	(2)	(2)	(2)	(8)	(8)	1	1
content	**255**	**CS7**	**P**	id	ct	pres	temp_INS	temp_OUT	gps_X	gps_Y	xor	**255**

where:

Id	**the same 4 bytes as received from the groundstation**
Ct	**describes the athmospheric content of the data frame** the byte ct is dechiffered as ct=64; ct+=1*hasPres; ct+=2*hasTemp_INS; …, where hasPres, hasTemp_INS, … are either 0 or 1.
Pres	**holds the analog to digital port content of the pressure sensor** if ADC represents the port content, then the byte pres is computed as pres=ADC&127; ADC<<=7; pres=ADC&127
temp_INS	**holds the port content of the inside temperature sensor** if ADC represents the port content, then the byte temp_INS is computed as temp_INS=ADC&127; ADC<<=7; temp_INS =ADC&127
temp_OUT	**holds the port content of the outside temperature sensor** analogous to temp_INS
gps_X	**holds the gps_X coordinate as string** since the GPS receiver gives a string as "9.9723123", we send the first 8 characters of this string
gps_Y	**holds the gps_Y coordinate as string** analogous to gps_Y
Xor	**checksum of all bytes including final 255** (the leading bit of xor should result in 0)

The character "P" represents "put". The frame is sent twice a second until Cansat receives a different request.

GS: One of the many identical frames from Cansat reaches eventually the Groundstation, which stores and visualizes the atmospheric data and requests the next frame by sending

byte(s)	1	3	1	4	1	1
content	**255**	**CS7**	**G**	id	xor	**255**

Since the groundstation is requesting the second frame, the index computes as

id=1 translating to b3=b2=b1=0, but b0=1.

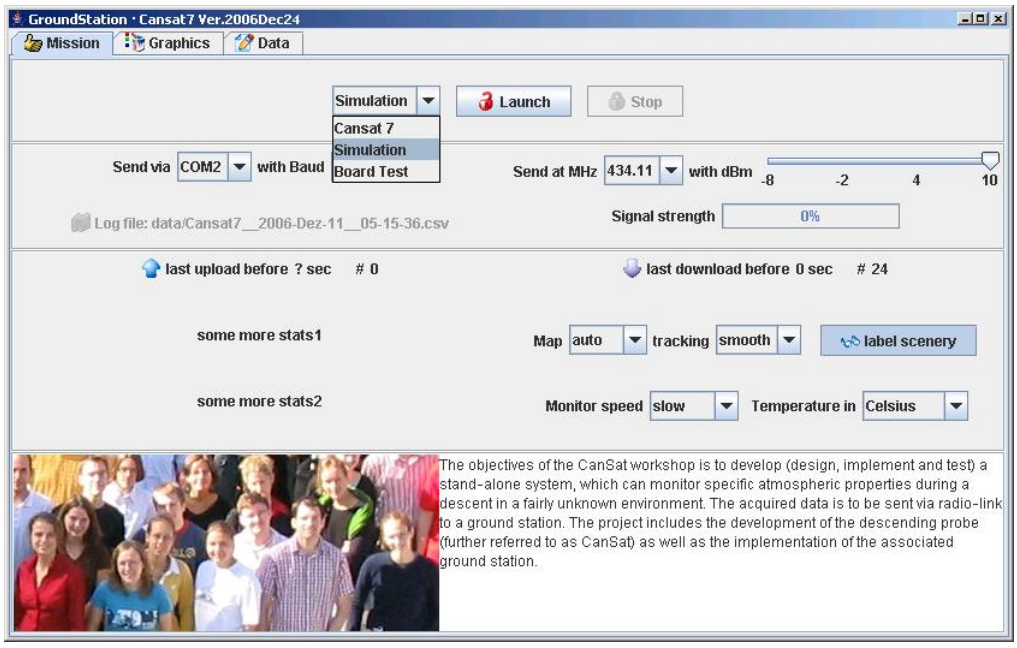
Again, this frame is sent until GS receives an appropriate answer.

CS: Eventually, the Cansat receives this frame. Now, the Cansat knows, that GS has received the atmospheric data with index 0, and frees that memory. Cansat answers with the data frame with index id=1, sending it every second until Cansat receives a different request.

This process continues in the obvious way.

Features:

- The protocol for testing and debugging comprises of a single mode for Groundstation (require data) and a single mode for the Cansat (provide data). Therefore, both implementations consists mainly of a single (infinite) loop.
- The link recovers from a temporary communication loss without difficulties, because there is no change of state/mode involved.
- Implementation does not extend more than two pages of code, both in Java and C.
- We avoid the characters 128-160 of the ASCII table, which cause ambiguities in Java.



Groundstation Software

The Groundstation software is complete, except for the implementation of the Cansat 7 protocol. In the previous milestone, we have reported about the principle class structure of the code. We have also provided screenshots.

In the following, we point out, what kind of changes/advancements have been accomplished, thereafter. Also, we append a paragraph on future work.

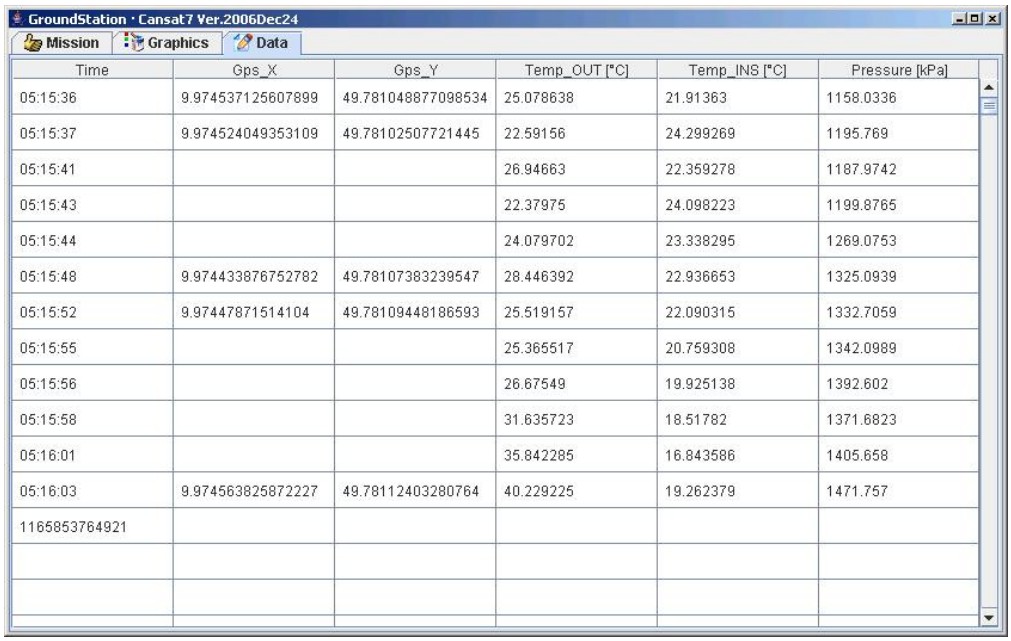
Changes/advancements:

As shown in the screenshot, the software realizes three “missions”:

1) The Cansat 7 mission, that communicates with the Cansat device.
2) A standalone simulation of athmospheric data input, to test and demonstrate the functionality of the components, without Cansat device active.
3) A Groundstation Circuit Board Test, which checks the setup of the transceiver chip and sends data, which is possibly received by another groundstation.

The Cansat 7 mission

The “Cansat 7” mission is equivalent to the “Simulation” mission, except that the athmospheric data is acquired from the Cansat device. Since the implementation of the “Simulation” mission is complete, the only remainder is the implementation of the communication protocol.



Time	Gps_X	Gps_Y	Temp_OUT [°C]	Temp_INS [°C]	Pressure [kPa]
05:15:36	9.974537125607899	49.781048877098534	25.078638	21.91363	1158.0336
05:15:37	9.974524049353109	49.78102507721445	22.59156	24.299269	1195.769
05:15:41			26.94663	22.359278	1187.9742
05:15:43			22.37975	24.098223	1199.8765
05:15:44			24.079702	23.338295	1269.0753
05:15:48	9.974433876752782	49.78107383239547	28.446392	22.936653	1325.0939
05:15:52	9.97447871514104	49.78109448186593	25.519157	22.090315	1332.7059
05:15:55			25.365517	20.759308	1342.0989
05:15:56			26.67549	19.925138	1392.602
05:15:58			31.635723	18.51782	1371.6823
05:16:01			35.842285	16.843586	1405.658
05:16:03	9.974563825872227	49.78112403280764	40.229225	19.262379	1471.757
1165853764921					

The Simulation mission

The simulation mission has been implemented several weeks ago. Recent changes are

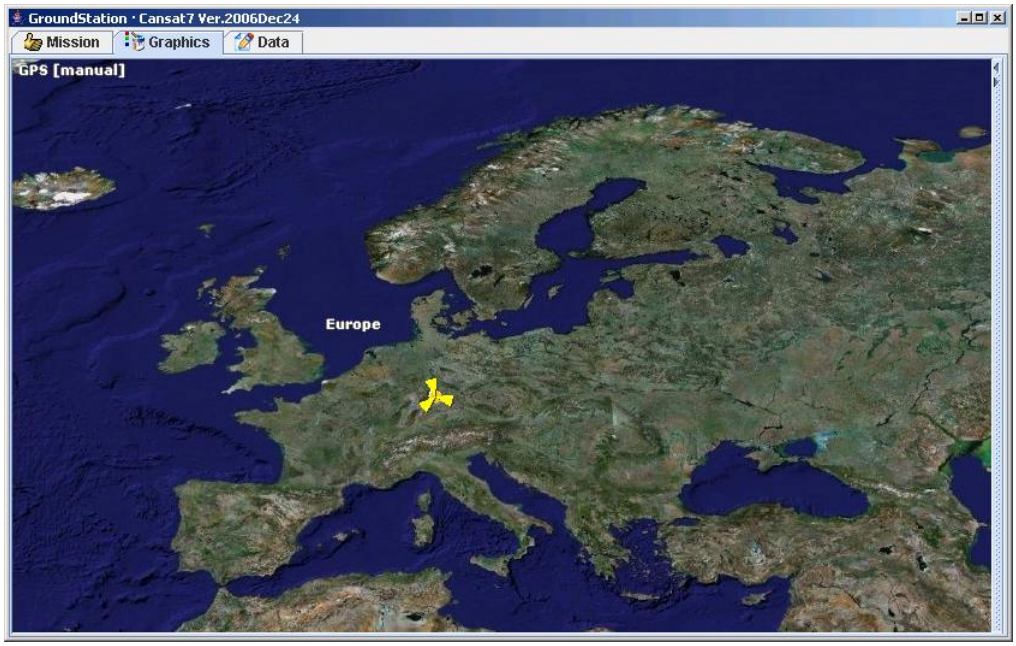
- the display of 2 temperature curves in the monitor window
- we automized the map to keep the Cansat in the center of the GPS display
- as an option, we provide labels of the building of the campus

The Board Test

We give a detailed description of the (already performed) Groundstation Circuit Board Test in Page 14. We have been collaborating with Cansat Group 6. The board test is tracked in the data table, as shown in screenshot.

General changes

We hope to have improved on the user interface and the principle layout. For instance, we have increased the font size and tabulator size in the table of data frames.



Code of interest:

1) The following exerpt of our program performs the *listening* and *read* operation of the serial port. Obviously, this is a crucial part!

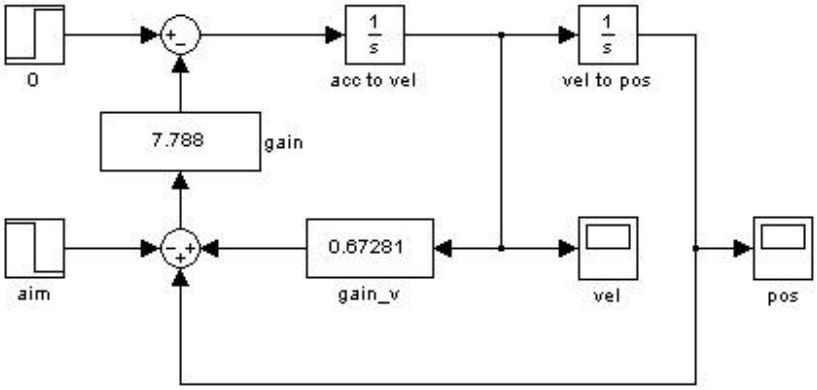
```
volatile char[] characters;
volatile int tail=0;
volatile int head=0;
public static final int BUFFER=1024;

public synchronized void serialEvent(SerialPortEvent mySerialPortEvent) {
  int c1;
  if (mySerialPortEvent.getEventType()==SerialPortEvent.DATA_AVAILABLE) {
    try {
      byte[] myByte=new byte[myInputStream.available()];
      myInputStream.read(myByte);
      for (c1=0;c1<myByte.length;c1++) {
        characters[(tail+c1)%BUFFER]=(char)(((int)myByte[c1]+256)&255);
      }
      tail=(tail+myByte.length)%BUFFER;
    } catch (IOException myIOException) {
      System.out.println("serial port input problem");
    }
  }
}

String readFlush() {
  String myString="";
  while (head!=tail) {
    myString+=characters[head++%BUFFER];
    head%=BUFFER;
  }
  return myString;
}
```

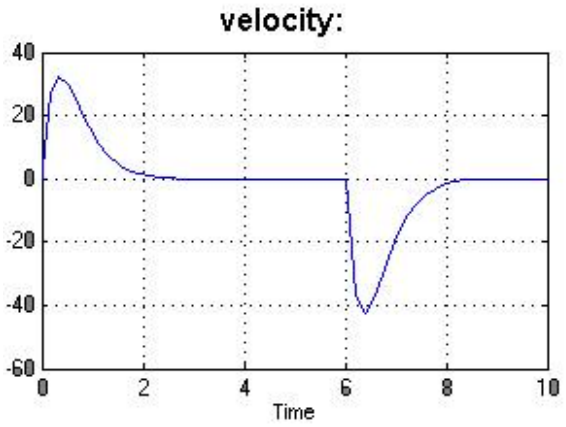
2) The next sequence of code calls the readFlush(). Since several messages might have arrived since the last read call, the overall input is decomposed into the simgle messages, which can be processed afterwards.

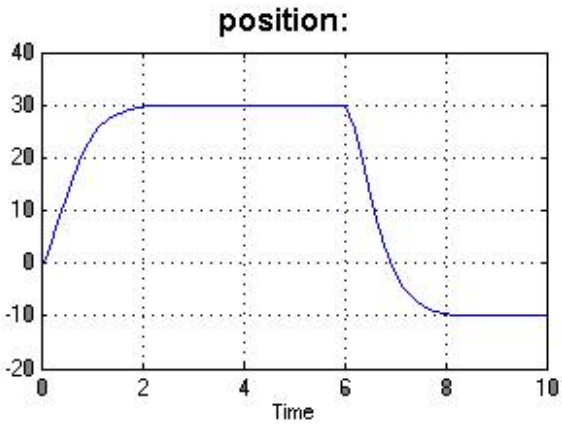
```
static final String DELIM=""+(char)255;
String myString;
StringTokenizer myStringTokenizer;
//...
  myTerminal.print("..."); // send command
  myThread.sleep(200);     // wait for answer
  // get answer:
  myStringTokenizer=new StringTokenizer(myTerminal.readFlush(),DELIM,false);
  // process all messages contained in answer:
  while (myStringTokenizer.hasMoreTokens()) {
    myString=myStringTokenizer.nextToken();
    //... process single message myString
  }
```

Future extensions:

- The implementation of the “debugging and testing”-protocol, as explained above. However, this needs to be done simultaneous to the development of the Cansat microcontroller code.
- In the next weeks, we will incorporate **a map of the launch site at Kiruna**. If not explicitly specified by the user, the software will autonomously decide, which map to choose for GPS tracking. The necessity of this action is clear: If our Cansat is launched in Kiruna, then the campus map of Würzburg is not sufficient to display the subsatellite point.
- We will implement **smooth tracking of the cansat location**. As the cansat gps coordinate varies over the course of the mission, the map to display the current position has to be moved along with the cansat subsatellite point. We have devised the following stable control loop that we implement in two dimensions. As shown by the plots of velocity and position, the subsatellite point is moved to the center of display after no more than 2 seconds.







Conclusions and future work

As explained over the course of the report, our group is well in the time schedule. All components are working and tested for basic functioning. What remains is some more refinement according to the specifications laid out in this report – for instance the rigid implementation of the sophisticated protocol.

Although our group is well in the time schedule, we do not switch to relaxed mode. We will stay focused and perform hard tests, which will put our cansat device to the test. This way our project will probably win the overall cansat group contest, because we will devise the most reliable communication philosophy and structure.

An important feature of our system is that if gps data is not available, the system does not yield functioning. This event will occur if the cansat is inside a building, within a tunnel, or underground. The remote device cansat will just send temperature and pressure sensor.

References

[ATmega128] 8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash, Atmel Corporation, 2004

[CP2102] Single-Chip USB to UART Bridge}, Silicon Laboratories, 2004

[FM75] Low Voltage 2-Wire Digital Temperature Sensor with Thermal Alarm, www.fairchildsemi.com

[GR213] HOLUX GR-213, GPS Receiver, User's guide, HOLUX Technology Inc., 2005

[Kr00] Guido Krueger: Go To Java 2, Handbuch der Java-Programmierung, 2. Auflage, Addison-Wesley, 2000

[JP7T] Fadil Beqiri: JP7-T Family GPS-Receiver Hardware description}, FALCOM GmbH, www.falcom.de

[MAX3232] 3.0V to 5.5V, Low-Power, up to 1Mbps, True RS-232, Transceivers Using Four External Capacitors, Maxim Integrated Products, 1999

[MPX4115A] Integrated Silicon Pressure Sensor for Manifold Absolute Pressure, Altimeter or Barometer Applications, Motorola, Inc., 2001, www.freescale.com

[RT433F4] Infoblatt 433MHz-FM-Mehrkanal-Transceiver, Ingenieurbuero fuer Elektronik, Wildpoldsried, 2004, www.funkmodul.com

[TS2940] 1A Ultra Low Dropout Fixed Positive Voltage Regulator, TSC, 2003

[WQ05] Wang Hankang, Li Quancheng, Jiang Hai: CDMA based distribution power net automatization and data transmission system}, Power Engineering Conference, 2005. IPEC 2005. The 7th International.