LEAN CUBE SAT REPORT

By:

STAR WALKERS TEAM

Contents

CODING SECTION

DHT Sensor

DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output.

The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and spits out a digital signal on the data pin (no analog input pins are needed). It's fairly simple to use but requires careful timing to grab data.

Figure 1_DHT11 Sensor

Technical Details

- Low cost
- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 20-80% humidity readings with 5% accuracy
- Good for 0-50°C temperature readings ±2°C accuracy
- No more than 1 Hz sampling rate (once every second)
- Body size 15.5mm x 12mm x 5.5mm
- 4 pins with 0.1" spacing

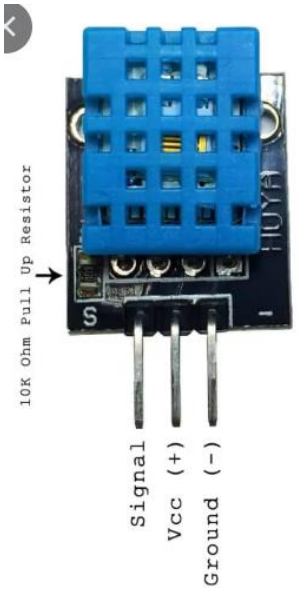

Figure 2_DHT11 Connection

BMP Sensor

It is a basic sensor that is designed specifically for measuring atmospheric pressure. As we travel from sea level to a mountain peak, the air pressure gets lower. That means by measuring the pressure we can determine the altitude. So, we can use this sensor as an Altimeter.

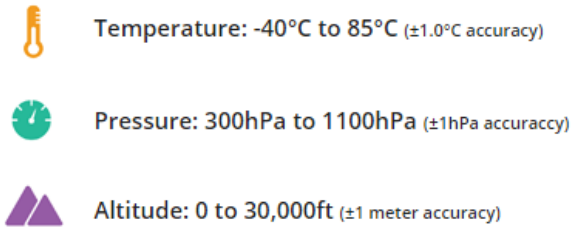

Figure 3_Range of BMP180 measurements

The module comes with an onboard LM6206 3.3V regulator and I2C Voltage Level Translator, so you can use it with a 3.3V or 5V logic microcontroller like Arduino without worry.

The module features a simple two-wire I2C interface that can be easily interfaced with any microcontroller of your choice. This module has a hardwired I2C address and is set to $0x77_{HEX}$.

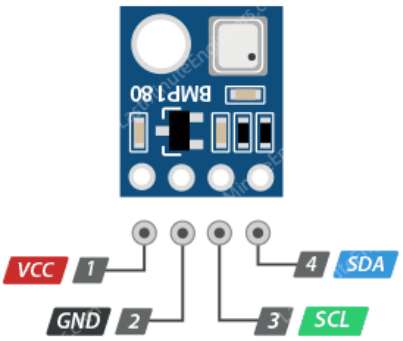

Figure 4_BMP180 Connection

Figure 5_BMP180 Sensor

MPU Sensor

The MPU6050 has both a gyroscope and an accelerometer, using which we can measure rotation along all three axes, static acceleration due to gravity, as well as motion, shock, or dynamic acceleration due to vibration.

The module uses the I2C interface for communication with Arduino. It supports two separate I2C addresses: $0x68_{HEX}$ and $0x69_{HEX}$. This allows two MPU6050s to be used on the same bus or to avoid address conflicts with another device on the bus.

A common unit of acceleration is G-force. One G-force here on Earth is 9.9 meters per second squared, which is the acceleration of gravity here. On other planets the figure is different. An accelerometer needs to take into account static acceleration like the force of gravity when making its measurements. The accelerometer used in the MPU-6050 is a triple-axis accelerometer, meaning that it senses acceleration on an X, Y ,and Z axis.

The type of gyroscope used in the MPU-6050 is a “Micro Electro Mechanical System” or MEMS gyroscope. It consists of three sensors, one per axis, that produce a voltage when they are rotated. This voltage is internally sampled using a 16-bit analog to digital converter.

Figure 7_MPU 6050 sensor

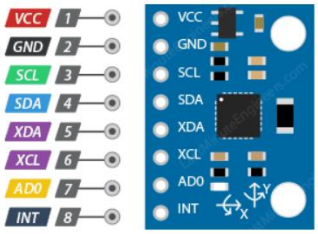

Figure 3_MPU 6050 connection

GPS Sensor

GPS is a system of 30+ navigation satellites orbiting the earth. We know where they are in space because they constantly transmit information about their position and current time to Earth in the form of radio signals.

A GPS receiver listens to these signals. Once the receiver calculates its distance from at least three GPS satellites, it can figure out where you are.

The data you are getting on the serial monitor are actually NMEA sentences.

NMEA is an acronym for National Marine Electronics Association. This is a standard message format for almost all GPS receivers.

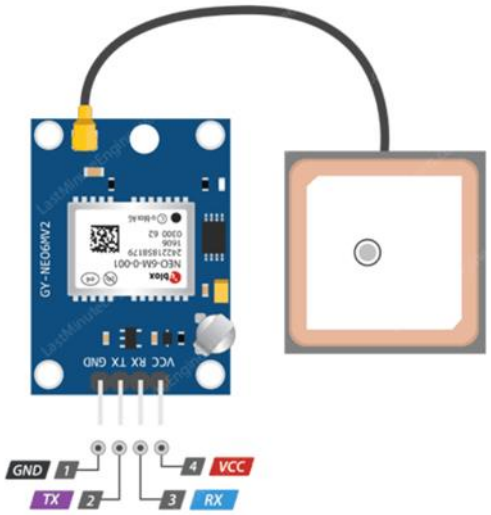

Figure 4_GPS Module Connection

$GPGGA, 123519, 4807.038, N, 01131.000, E, 1, 08, 0.9, 545.4, M, 46.9, M, , *47

$	Starting of NMEA sentence.
GPGGA	Global Positioning System Fix Data
123519	Current time in UTC – 12:35:19
4807.038,N	Latitude 48 deg 07.038′ N
01131.000,E	Longitude 11 deg 31.000′ E
1	GPS fix
08	Number of satellites being tracked
0.9	Horizontal dilution of position
545.4,M	Altitude in Meters (above mean sea level)
46.9,M	Height of geoid (mean sea level)
(empty field)	Time in seconds since last DGPS update
(empty field)	DGPS station ID number
*47	The checksum data, always begins with *

Figure 9_GPGGA NMEA Example

Figure 10_GPS Satellite System

Figure 11_GPS Module

SD Card Module

Before you insert the micro SD card into the module and hook it up to the Arduino, you must properly format the card.

Figure 5_SD Card Module

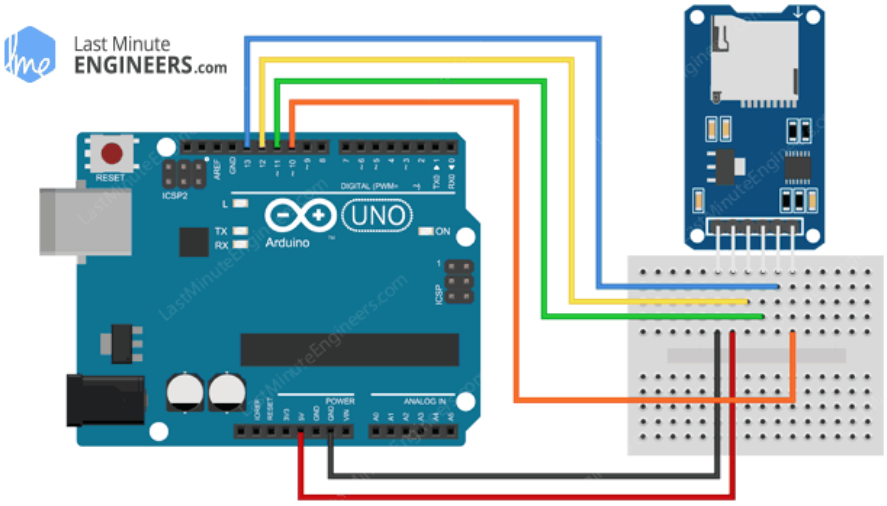

Figure 14_SD Card Module with Arduino

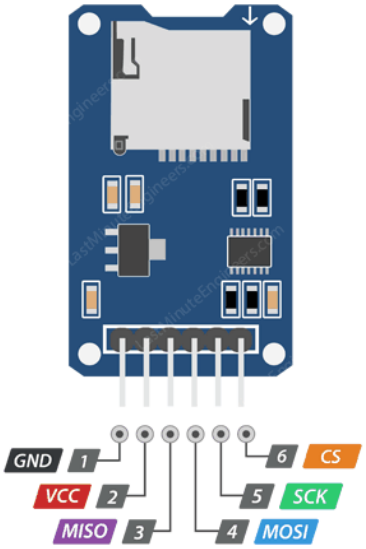

Figure 13_SD Card Module Connection

- The operating voltage of any standard micro SD Cards is 3.3 V. So we cannot directly connect it to circuits that use 5V logic. In fact, any voltages exceeding 3.6V will permanently damage the micro SD card. That's why; the module has an onboard ultra-low dropout regulator that will convert voltages from 3.3V – 6V down to ~3.3V. There's also a 74LVC125A chip on the module which converts the interface logic from 3.3V-5V to 3.3V. This is called logic level shifting. That means you can use this board to interact with both 3.3V and 5V microcontrollers like Arduino.
- every SD card module is based on a 'lower speed & less overhead' SPI mode that is easy for any microcontroller to use.

Main Code

```
#include <Wire.h>

//SD_Card //NOTE CS PIN IS CONNECTED TO DIGITAL PIN 10 IN ARDUINO
#include <SPI.h>
#include <SD.h>
#include <SD_Card_Module.h>
SD_Card_Module sd;

//BMP
#include <BMP180_Sensor.h>
BMP180_Sensor bmp;

//DHT // NOTE THE DATA PIN IS CONNECTED TO PIN NUMBER 2 IN ARDUINO
#include <DHT11_Sensor.h>
DHT11_Sensor dht;

//MPU
#include <MPU6050_Sensor.h>
MPU6050_Sensor mpu;

//GPS
#include <GPS_Sensor.h>
GPS_Sensor gps;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Wire.begin();
  Serial.print('*');

  //BMP
  bmp.GET_DATA_FROM_REGISTERS();

  //MPU6050
  mpu.start_communication();
  //GPS
  gps.Run();
}

void loop() {


//DHT
dht.calculate_parameters();// This function prints the following
("DHT:RH,TEMP")
Serial.print(';');
Serial.print("\n");
sd.store_data_DHT(dht.RH_int,dht.RH_dec,dht.Temp_int,dht.Temp_dec);// This
function prints ("done")
Serial.print("\n");

//BMP
bmp.calculate_parameters();// This function prints the following
("BMP:Pressure in pascal,Altitude")
Serial.print(';');
Serial.print("\n");
sd.store_data_BMP(bmp.P,bmp.A);// This function prints ("done")
Serial.print("\n");
```

```
//MPU
mpu.calculate_parameters();// This function prints the following
("MPU:AccX,AccY,AccZ,gyroAngleX,gyroAngleY,gyroAngleZ")
Serial.print(';');
Serial.print("\n");
sd.store_data_MPU(mpu.AccX,mpu.AccY,mpu.AccY,mpu.gyroAngleX,mpu.gyroAngleY,
mpu.gyroAngleZ);// This function prints ("done")
Serial.print("\n");
}
```

Second Arduino Code

```
void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
if (Serial.available() )
{
  Serial.print(Serial.readStringUntil("\n"));
}
}
```

BMP180 Code

```
#include <Wire.h>
class BMP180_Sensor
{
private:
#define BMP 0x77
#define AC1_REG 0xAA
#define AC2_REG 0xAC
#define AC3_REG 0xAE
#define AC4_REG 0xB0
#define AC5_REG 0xB2
#define AC6_REG 0xB4
#define BB1_REG 0xB6
#define BB2_REG 0xB8
#define MB_REG 0xBA
#define MC_REG 0xBC
#define MD_REG 0xBE
#define CONTROL 0xF4
#define TEMP 0x2E
#define PRESSURE 0x34
#define UT_REG 0xF6
short AC1;
short AC2;
short AC3;
unsigned short AC4;
unsigned short AC5;
unsigned short AC6;
short BB1;
short BB2;
short MB;
short MC;
short MD;
const unsigned char OSS=0;
long UT;
long UP;
long T;
const float PO=101500.0;


public:
long P;
float A;
int READ_REGISTER(unsigned char Register_Name)
{
  Wire.beginTransmission(BMP);
  Wire.write(Register_Name);
  Wire.endTransmission();
  Wire.requestFrom(BMP,2);
  if(Wire.available()==2)
  {
    short data_1=Wire.read();
    short data_2=Wire.read();
    short Convert=data_1<<8;
    short DATA_OUT=Convert|data_2;
    return DATA_OUT;
  }
}
```

```
void GET_DATA_FROM_REGISTERS()
{
  AC1=READ_REGISTER(AC1_REG);
  AC2=READ_REGISTER(AC2_REG);
  AC3=READ_REGISTER(AC3_REG);
  AC4=READ_REGISTER(AC4_REG);
  AC5=READ_REGISTER(AC5_REG);
  AC6=READ_REGISTER(AC6_REG);
  BB1=READ_REGISTER(BB1_REG);
  BB2=READ_REGISTER(BB2_REG);
  MB=READ_REGISTER(MB_REG);
  MC=READ_REGISTER(MC_REG);
  MD=READ_REGISTER(MD_REG);
}
void calculate_parameters()
{
  Wire.beginTransmission(BMP);
  Wire.write(CONTROL);
  Wire.write(TEMP);
  Wire.endTransmission();
  delay(5);
  UT=READ_REGISTER(UT_REG);
  Wire.beginTransmission(BMP);
  Wire.write(CONTROL);
  Wire.write(PRESSURE+(OSS<<6));
  Wire.endTransmission();
  delay(5);
  Wire.beginTransmission(BMP);
  Wire.write(UT_REG);
  Wire.endTransmission();
  Wire.requestFrom(BMP,3);
  if(Wire.available()==3)
  {
    unsigned long UP1=Wire.read();
    unsigned long UP2=Wire.read();
    unsigned long UP3=Wire.read();
    UP=((UP1<<16) + (UP2<<8) | (UP3)) >> (8-OSS);
  }
  long X1 = (UT-AC6)*AC5/(pow(2,15));
  long X2 = MC* pow(2,11)/(X1+MD);
  long B5 = X1+X2;
  T = (long)(B5+8) /pow(2,4);
  long B6 = B5-4000;
  X1 = (BB2 * (B6 * B6/pow(2,12))/pow(2,11));
  X2 = AC2 * B6/pow(2,11);
  long X3 = X1 + X2;
  long B3 = (((AC1*4 + X3)<<OSS) + 2)/4;
  X1 = AC3 * B6/pow(2,13);
  X2 = BB1 * (B6 * B6/pow(2,12))/pow(2,16);
  X3 = ((X1 + X2) + 2)/pow(2,2);
  unsigned long B4 = AC4 * (unsigned long)(X3 + 32768)/pow(2,15);
  unsigned long B7 = (UP - (unsigned long) B3) * (50000>>OSS);
  if (B7 < 0x80000000)
  {
    P = (B7*2)/B4;
  }
  else
  {
    P = (B7/B4)*2;
  }
  X1 = (P/pow(2,8)) * (P/pow(2,8));
```

```
  X1= (X1 * 3038)/pow(2,16);
  X2 = (-7357 * P)/pow(2,16);
  P += (X1 + X2 + 3791)/pow(2,4);
  P =P - 47670;
  A=44330 * (1.0 - pow( P/PO, 0.1903));
  Serial.print("BMP:");Serial.print(P); Serial.print(","); Serial.print(A);
}
};
```

DHT11 Code

```
class DHT11_Sensor {
private:
#define pin 2

public:
unsigned char RH_int =0,RH_dec=0,Temp_int=0,Temp_dec=0,sum=0;
int expectPulse(bool lvl)
{
  int count = 0;
  while (digitalRead(pin)== lvl) count++; // lvl while be once high and
then low , Im gonna compare between both of them to know if the signal fron
sensor is 0 or 1
  return count ; // if the peroide of high is bigger than low that means
it's 1
  }
  void calculate_parameters()
  {

  unsigned char pulse[80]; //array for pulses
pinMode (pin,OUTPUT);
digitalWrite (pin,HIGH );
delay (1000);
digitalWrite (pin,LOW);
delay (18);
digitalWrite (pin, HIGH);
delayMicroseconds (40);
pinMode (pin,INPUT);
if (digitalRead (pin)== LOW)
{
  while (digitalRead (pin)== LOW)
  {
    //delayMicroseconds(80);
    }
    while (digitalRead (pin)== HIGH)
  {
    //delayMicroseconds(80);
```

```
    }
  } else {
    //Serial.println("not responding");
    return ;
    }
    for (int i=0; i<80 ; i++)
      pulse[i]= expectPulse(i%2);
      // i=2 ,lvl=0 , expectpulse= count= 20 = pulse[2]
      // i=3 , lvl =1 , expectpulse = count = 30 = pulse[3]
      /// pulse[80] = {20,30,50,10,.....}

    for (int i=0 ; i<40; i++)
    {
      unsigned char lowCycle = pulse[i*2];
      unsigned char highCycle = pulse[i*2 +1];
      // shifting
      if (i<8)
      {
        RH_int <<=1;
        // 1 = 0001 ////// rh_int = 00000000
        // rh_int = 00000001
        if (highCycle > lowCycle ) RH_int |=1;
        ///
        }
      if (i<16)
      {
        RH_dec <<=1;
        if (highCycle > lowCycle ) RH_dec |=1;
        }
      if (i<24)
      {
        Temp_int <<=1;
        if (highCycle > lowCycle ) Temp_int |=1;
        }
      if (i<32)
      {
        Temp_dec <<=1;
        if (highCycle > lowCycle ) Temp_dec |= 1;
        }
      if (i<40)
      {
        sum <<=1;
        if (highCycle > lowCycle ) sum |=1;
      }
    }
      if (sum != RH_int+RH_dec+Temp_int+Temp_dec)
      {
        Serial.println("Not responding:");
        } else {

          Serial.print("DHT11:");
          Serial.print(RH_int); Serial.print(".") ; Serial.print(RH_dec);
          Serial.print(',');
          Serial.print(Temp_int); Serial.print(".") ;
Serial.print(Temp_dec);

          }
  }

};
```

MPU Code

```
class MPU6050_Sensor{
 private:
#include <Wire.h>
const int MPU = 0x68; // MPU6050 I2C address
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY;
//float roll, pitch, yaw;
//float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ;
float elapsedTime, currentTime, previousTime;
int c = 0;

public:
float AccX, AccY, AccZ;
float gyroAngleX, gyroAngleY, gyroAngleZ;
void start_communication()
{
  Wire.beginTransmission(MPU);       // Start communication with MPU6050 //
MPU=0x68
  Wire.write(0x6B);                  // Talk to the register 6B
  Wire.write(0x00);                  // Make reset - place a 0 into the 6B
register
  Wire.endTransmission(true);        //end the transmission
}

void calculate_parameters()
{
  Wire.beginTransmission(MPU);
  Wire.write(0x3B); // Start with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 6 registers total, each axis
value is stored in 2 registers
  //For a range of +-2g, we need to divide the raw values by 16384,
according to the datasheet
  AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // X-axis value
  AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // Y-axis value
  AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // Z-axis value
  // Calculating Roll and Pitch from the accelerometer data
// accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI)
- 0.58; // AccErrorX ~(0.58) See the calculate_IMU_error()custom function
for more details
 // accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180
/ PI) + 1.58; // AccErrorY ~(-1.58)
  // === Read gyroscope data === //
  previousTime = currentTime;        // Previous time is stored before the
actual time read
  currentTime = millis();            // Current time actual time read
  elapsedTime = (currentTime - previousTime) / 1000; // Divide by 1000 to
get seconds
  Wire.beginTransmission(MPU);
  Wire.write(0x43); // Gyro data first register address 0x43
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 4 registers total, each axis
value is stored in 2 registers
  GyroX = (Wire.read() << 8 | Wire.read()) / 131.0; // For a 250deg/s range
we have to divide first the raw value by 131.0, according to the datasheet
  GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
  GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;
  // Correct the outputs with the calculated error values
  GyroX = GyroX + 0.56; // GyroErrorX ~(-0.56)
```

```
  GyroY = GyroY - 2; // GyroErrorY ~(2)
  GyroZ = GyroZ + 0.79; // GyroErrorZ ~ (-0.8)
  // Currently the raw values are in degrees per seconds, deg/s, so we need
to multiply by sendonds (s) to get the angle in degrees
  gyroAngleX = gyroAngleX + GyroX * elapsedTime; // deg/s * s = deg
  gyroAngleY = gyroAngleY + GyroY * elapsedTime;
  //yaw =  yaw + GyroZ * elapsedTime;
  // Complementary filter - combine acceleromter and gyro angle values
 // roll = 0.96 * gyroAngleX + 0.04 * accAngleX;
  //pitch = 0.96 * gyroAngleY + 0.04 * accAngleY;

  // Print the values on the serial monitor
Serial.print("MPU:");Serial.print(AccX);Serial.print(',');Serial.print(AccY
);Serial.print(',');Serial.print(AccZ);Serial.print(',');Serial.print(gyroA
ngleX);Serial.print(',');Serial.print(gyroAngleY);Serial.print(',');Serial.
print(gyroAngleZ);
}
};
```

SD Module Code

```
class SD_Card_Module{

private:

const int chipSelect = 10;


public:



void store_data_DHT(char x, char y, char z, char L)
{  String dataString_DHT = "";
   dataString_DHT += String(x);
   dataString_DHT += ".";
   dataString_DHT += String(y);
   dataString_DHT += ",";
   dataString_DHT += String(z);
   dataString_DHT += ".";
   dataString_DHT += String(L);
   File data_DHT = SD.open("data_DHT.txt", FILE_WRITE);
   if (data_DHT) {
    data_DHT.println(dataString_DHT);
    data_DHT.close();
```

```
   }
   Serial.print("done:connected");
}

void store_data_BMP(long a, float b)
{
  String dataString_BMP = "";
   dataString_BMP += String(a);
   dataString_BMP += ",";
   dataString_BMP += String(b);
    File data_BMP = SD.open("data_BMP.txt", FILE_WRITE);
   if (data_BMP) {
    data_BMP.println(dataString_BMP);
    data_BMP.close();
   }
   Serial.print("done:connected");
}
void store_data_MPU(float q, float w, float e, float r, float t, float u)
{
  String dataString_MPU = "";
   dataString_MPU += String(q);
   dataString_MPU += ",";
   dataString_MPU += String(w);
   dataString_MPU += ",";
   dataString_MPU += String(e);
   dataString_MPU += ",";
   dataString_MPU += String(r);
   dataString_MPU += ",";
   dataString_MPU += String(t);
   dataString_MPU += ",";
   dataString_MPU += String(u);
   File data_MPU = SD.open("data_MPU.txt", FILE_WRITE);
   if (data_MPU) {
    data_MPU.println(dataString_MPU);
    data_MPU.close();
   }
   Serial.print("done:connected");
}
};
```

GPS Code

```
class GPS_Sensor{
    private:
char MID[6];
char UTC[11];
char LAT[10];
char NSIND[2];
char LONG[11];
char EWIND[2];
char POSFIX[2];
char SAT[3];
char ALTITUDE[7];

public:

void Run(){
    if (Serial.available())
    {

   if (Serial.read() == '$')
 {
 Serial.readBytesUntil(',' , MID, 6);
 MID[5] = '\0';
 if (MID[2] == 'G' && MID[3] == 'G' && MID[4] == 'A')
 {
   Serial.readBytesUntil(',' , UTC, 11);
   Serial.readBytesUntil(',' , LAT, 10);
   Serial.readBytesUntil(',' , NSIND, 2);
   Serial.readBytesUntil(',' , LONG, 11);
   Serial.readBytesUntil(',' , EWIND, 2);
   Serial.readBytesUntil(',' , POSFIX, 2);
   Serial.readBytesUntil(',' , SAT, 3);
   Serial.readBytesUntil(',' , ALTITUDE, 3);
   Serial.readBytesUntil(',' , ALTITUDE, 7);

 UTC[10]='\0';
 LAT[9]='\0';
 NSIND[1]='\0';
 LONG[10]='\0';
 EWIND[1]='\0';
 POSFIX[1]='\0';
 SAT[2]='\0';
 ALTITUDE[6]='\0';
 Serial.print("ID: ");
Serial.println(MID);
Serial.print("UTC: ");
Serial.println(UTC);
Serial.print("Latitude: ");
Serial.println(LAT);
Serial.print("NS: ");
Serial.println(NSIND);
Serial.print("Longitude: ");
Serial.println(LONG);
Serial.print("EW: ");
Serial.println(EWIND);
Serial.print("Position Fix: ");
Serial.println(POSFIX);
Serial.print("Satellite ");
Serial.println(SAT);
```

```
Serial.print("Altitude: ");
Serial.println(ALTITUDE);
}

 }

 }
    }

};
```

STRUCTURAL DESIGN SECTION

Software implementation section

First we started drawing on a paper the parts' shapes and concept that we used in our project's structure with more details of length, width and thickness measurements.

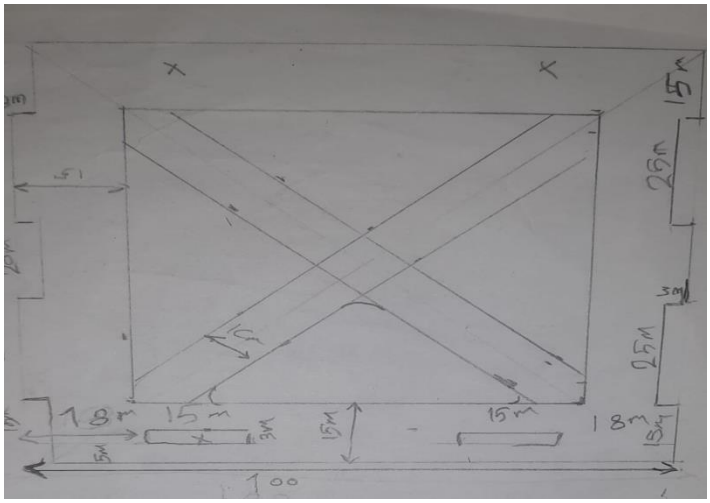

Figure 15_hand-made drawing to the wall of the lean CubeSat

We then have opened a new part in solid work, then we start a sketch and draw our parts specifying it with smart dimensions tool.

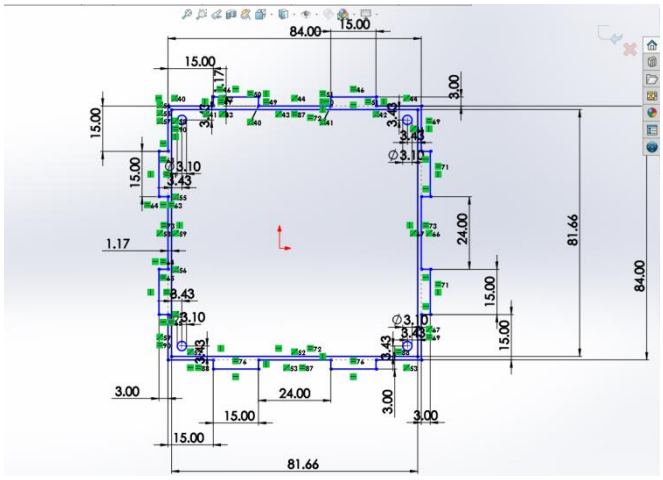

Figure 16_sketch for both base and roof of the cube in solidwork

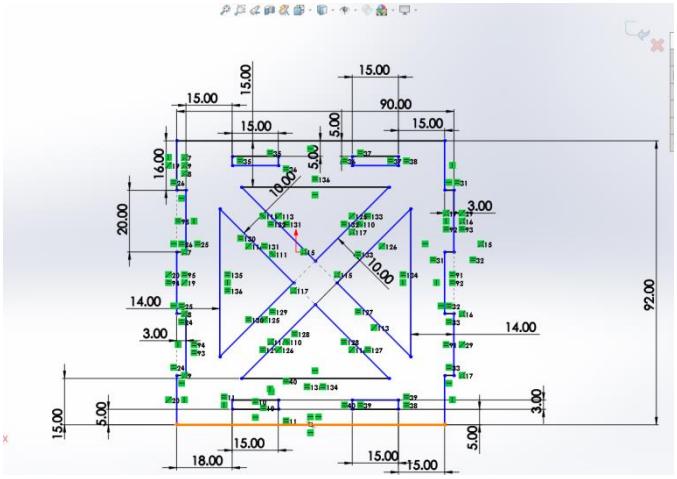

Figure 17_sketch for all 4 walls of the cube in solidwork

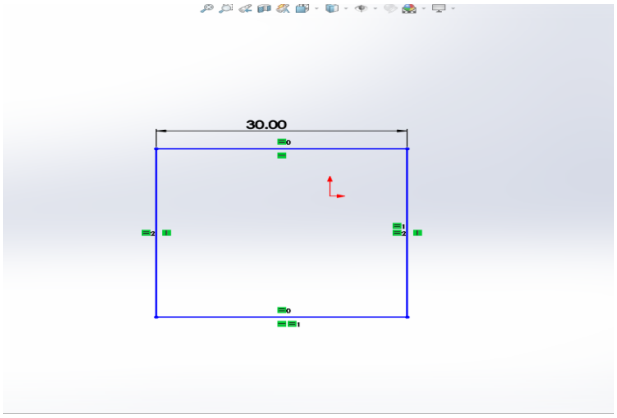

Figure 18_sketch for al1 8 corners of the cube

After finishing our sketch we choose Extrude-boss feature to make it a body with a thickness not just lines.

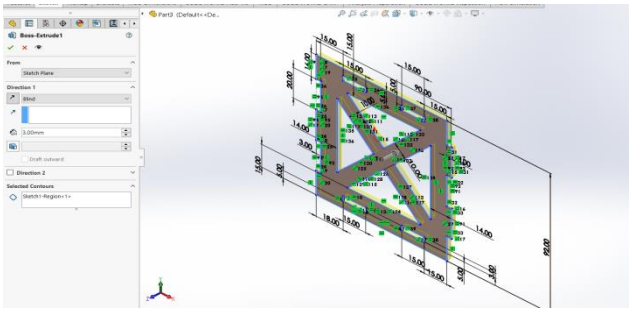

Figure 19_Extrode -boss feature to the wall of the cube

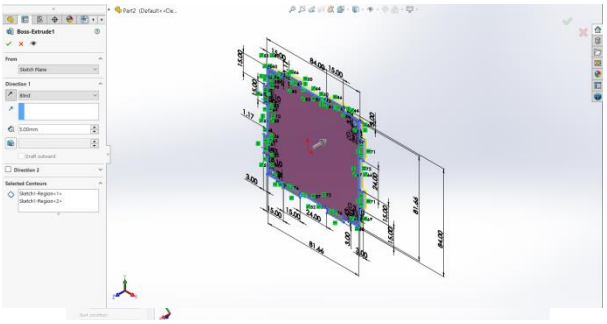

Figure 20_Extrude-boss feature for both the base and roof of the cube

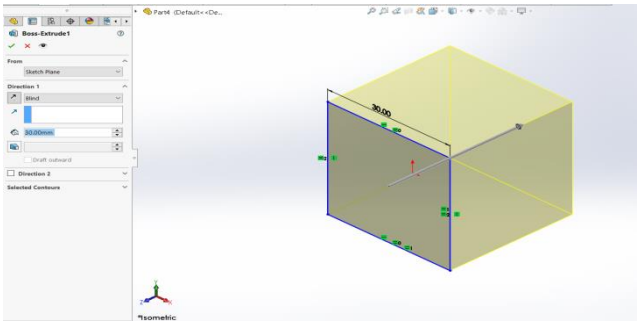

Figure 21_Extrude-boss feature to the corner we made for the cube

If there are any more parts we need to cut out from our major part we used for it Extrude-cut feature

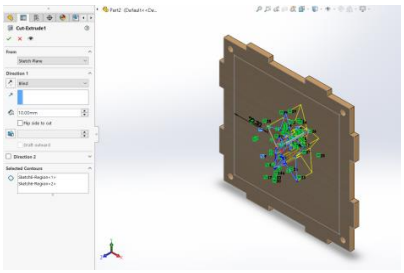

Figure 22_Extrude-cut feature to the roof

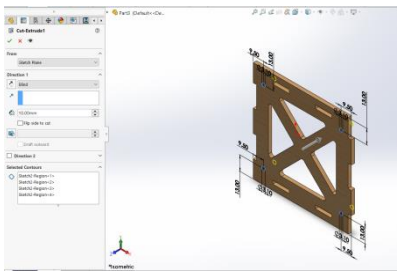

Figure 23_Extrude-cut to the wall to make screw's hole

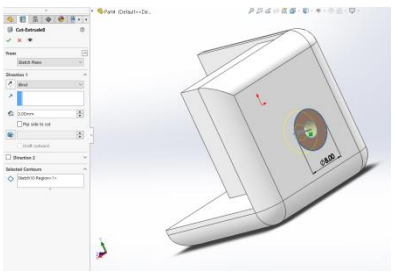

Figure 24_Extrude-cut to make a hole in the corner for the head of screw

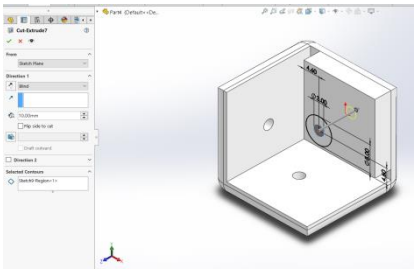

Figure 25_Extrude-cut to the corner to make screw's hole

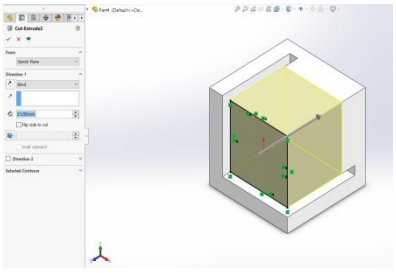

Figure 26_Extrude-cut in the corner to make it well installed in the structure

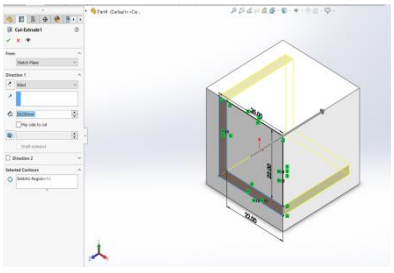

Figure 27_Extrude-cut in the corner to make it well installed in the structure

We had needed to use Fillet feature to make the shape of some of our parts more aesthetic, not only just intersecting lines with angles, we needed to feed the design with curved shapes.

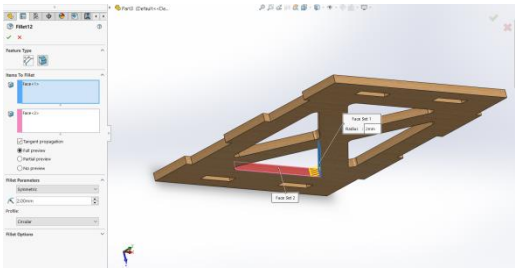

Figure 28_ fillet feature to internal corner of the wall

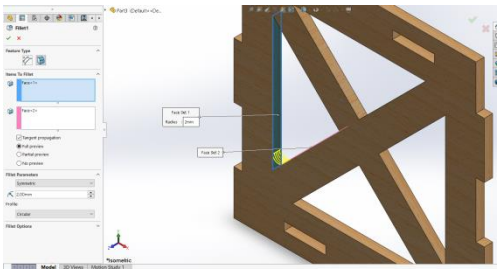

Figure 29_ fillet feature to another intenal corner in the wall

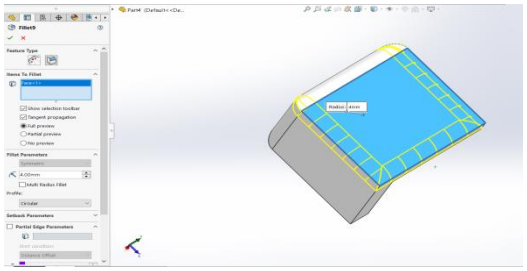

Figure 30_ fillet feature to the plastic corner

Then we struggled to make adjustments to the dimensions of parts used in our structure to make both the length, width and height equal only to 100 mm

After we finished the adjustments we thought about how to install very well the PCBs bearing project's sensors in our structure. Then we designed parts that would bear the PCBs installing and connecting them with the structure, then we were surprised that there are some things similar to those parts called "spacers" sold only and exclusively at some electronics stores located in Cairo

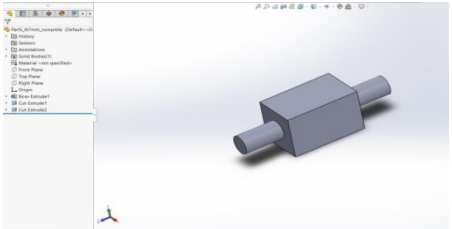

Figure 31_Our design for the part which we will need to install PCBs

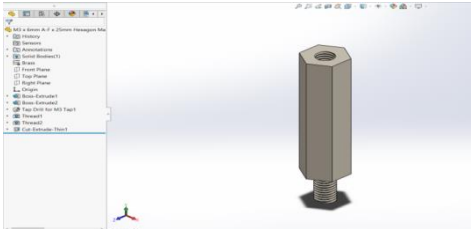

Figure 32_Solidwork design for the spacer

We again encountered that when we decided to design a battery bed, we knew that there is also something sold ready to install the battery with the structure called “battery holder”.

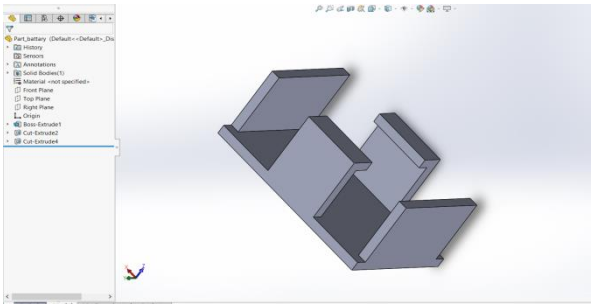

Figure 33_Our designing for a battery bed

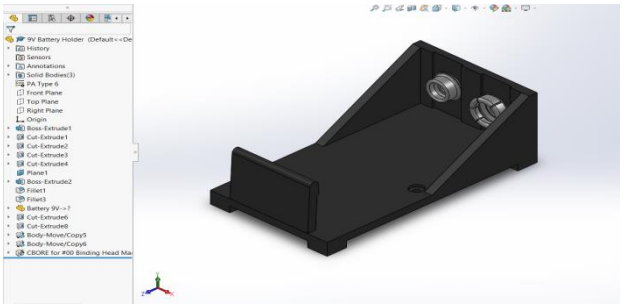

Figure 34_A solidwork design for the battery holder

After all this, we were faced with the fact that for someone who made 2 PCBs carrying sensors with high heights mounted on headers with other heights that the sizes are not suitable with small heights such as 100 mm (take the thickness of structure in your considering) , it could have been solved that the design of the PCBs is being modified so that the spaces are exploited very well, but unfortunately it was too late taking into account the time taken from the PCBS design team to make another design, the manufacturing time of the PCBs and the time when it is supposed to finish the project delivery (our lean cube sat).

With this, we were somewhat forced to move forward with our initial design to catch up with the completion time of the delivery of our project, taking into account that it is possible that some sensors do not have the appropriate height to install them or to put them in their correct places on the PCBs.

But we were continuing to meet the requirements that both the height, width and length of the cube should not exceed 100 mm and that the structure should be so durable that if a drop test is performed on it with its weight from a height of 2 meters, no very harmful fractures occur.

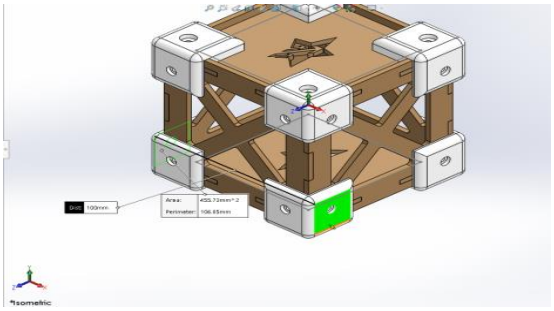

Figure 35_One dimension of our structure

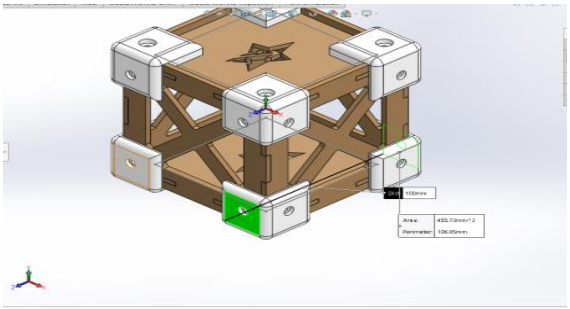

Figure 36_Second dimension of our structure

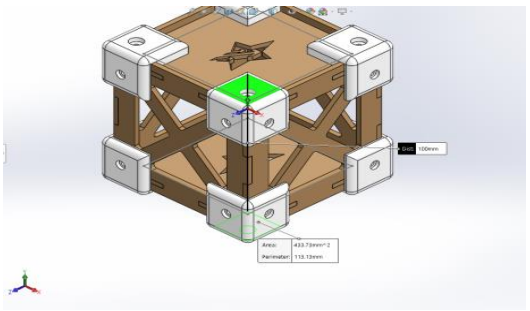

Figure 37_Third dimension of our structure

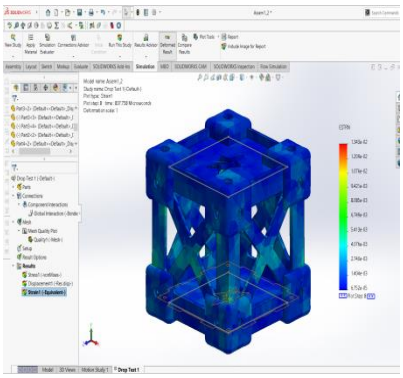

Figure 38_Strain analysis to the structure

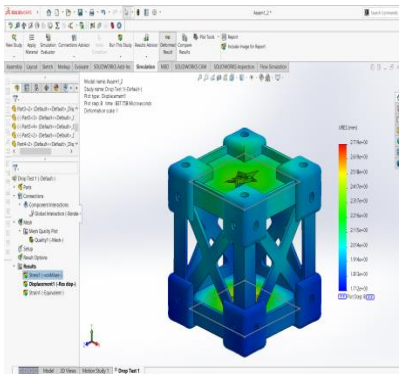

Figure 39_Distance analysis to the structure

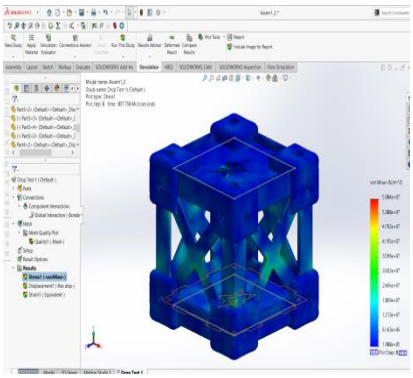

Figure 40_Stress analysis to the structure

Hardware implementation section

First, we converted the solid work files of the parts that will be manufactured to fit both the laser cutting machine and the 3D- printing machine (DXF files for laser cutting machine and STL files for 3D printing machine).

Then we handed over those files to the responsible engineer and specialist in the manufacturing issue, waiting for them to be received, to install them and take the appropriate spacers' sizes and heights to buy them from the dedicated stores that sell them

After receiving those pieces and parts, we prepared and took some suitable sizes for certain spacers to install the PCBs on them by taking into account both the height of the battery, the thickness of the battery holder, the thickness of the PCBs, the maximum height of the Arduino Uno and the maximum height of the sensors mounted on the headers and then we went to buy these spacers

Figure 41_Different sizes of spacers

After all this we have made preparations for connecting and installing all the parts and components of our lean cube sat to each other

With all our regrets, but the time was too tight to test and make sure the integrity of the welding on the PCBs and also install all the screws in all the pieces.

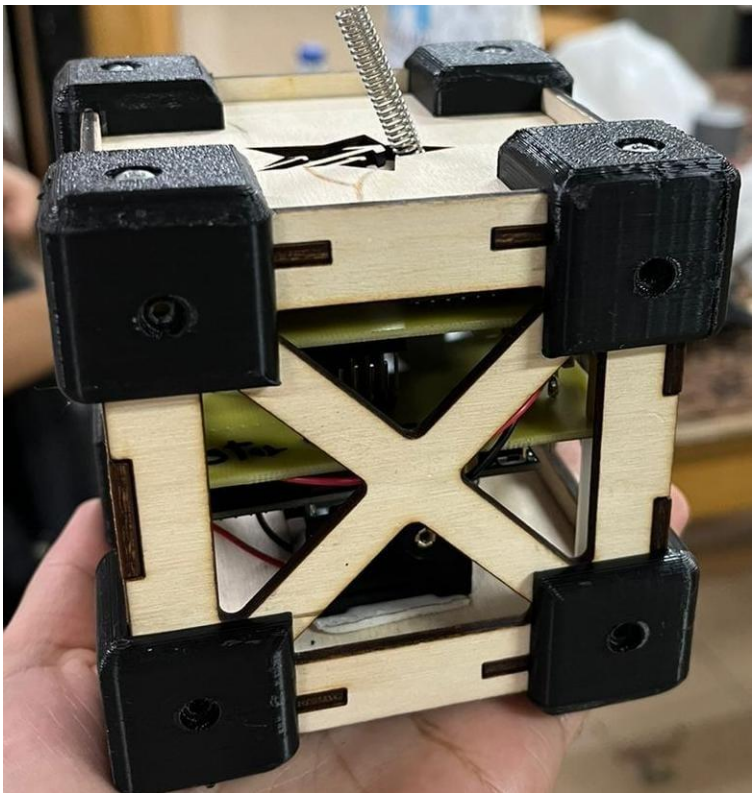

Figure 42_The final shape for our lean cube sat

PCBs DESIGN SECTION

PCB Design

- We have two PCB boards .
- There are headers to link between both of them .
- PCB contains (Arduino uno pads , sensors pads : MPU6050 , DHT11,BMP180,GPS NEO6M, RF , Micro SD card) , female headers and power circuit

1) First PCB :

- it contains power circuit , Arduino uno pads , MPUU6050 and female headers

Component	Hole size	Width of pads	Tracks size
Dualling-line packages(D	1mm	1.5mm	1mm

1.1 Arduino uno : microcontroller

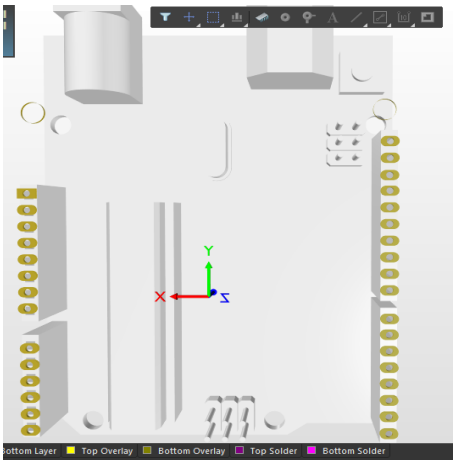

Figure 43_Arduino PCB-Lib

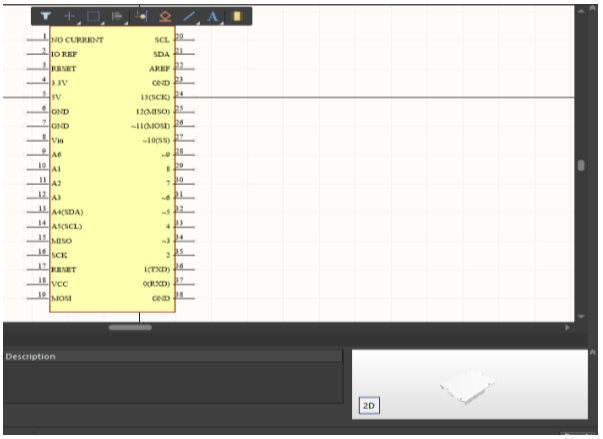

Figure 44_Arduino SCH-Lib

- 1.2 MPU6050 : to measure acceleration and gyroscope

VCC	SCL	SDA
3.3V	A5 in Arduino	A4 in Arduino

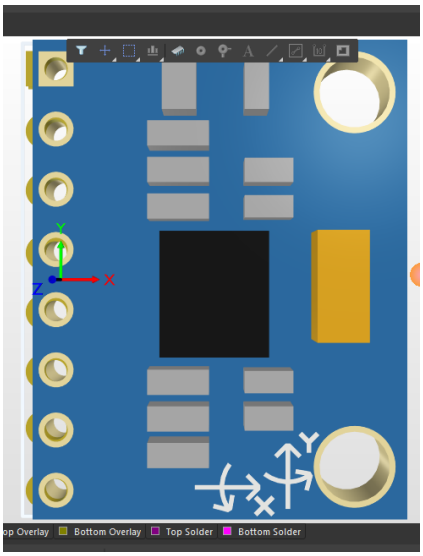

Figure 45_MPU6050 PCB-Lib

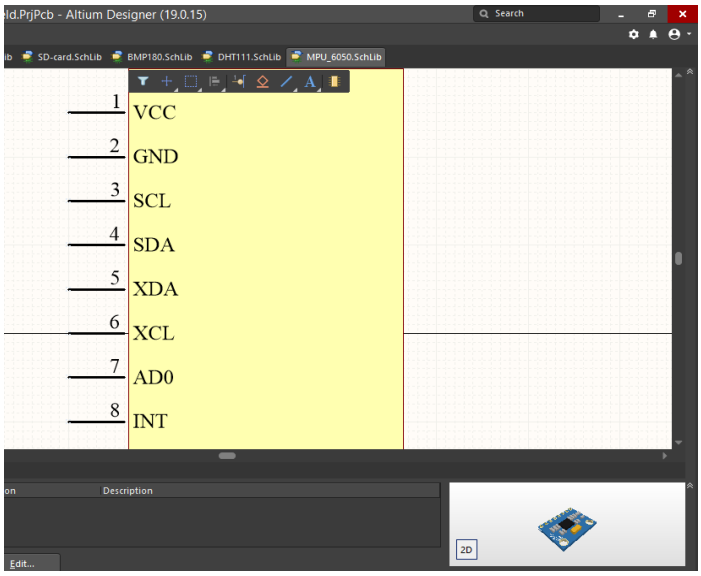

Figure 46_MPU6050 SCH-Lib

- 1.3 power circuit :
- (2 capacitors , led , resistor , Battery 9V, header 3 H and switch)
- -It converts 9v into 5v .

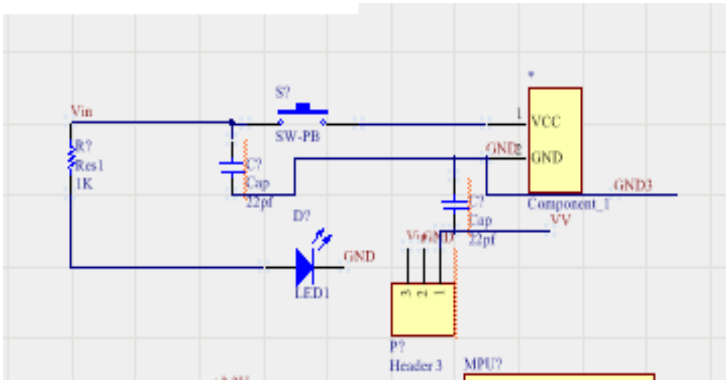

Figure 47_Power Circuit

- 1.4 Headers : (SCL , SDA , GND , +3.3V , RX , +5V , TX , DATA , MISO , MOSI , SCK , CS)

female headers .

-link between PCB boards .

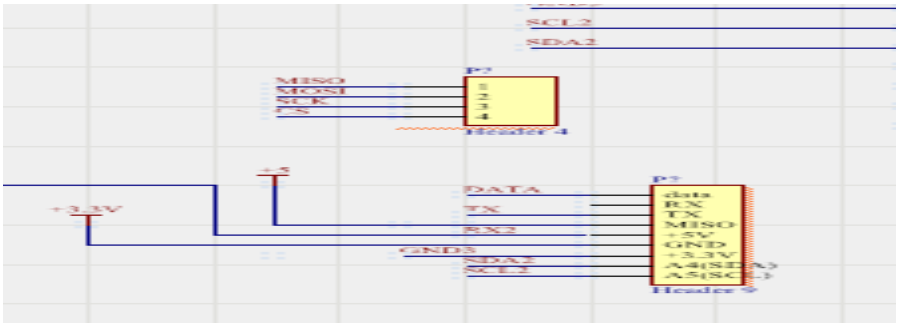

Figure 48_Headers

1.5 SCHEMATICS :

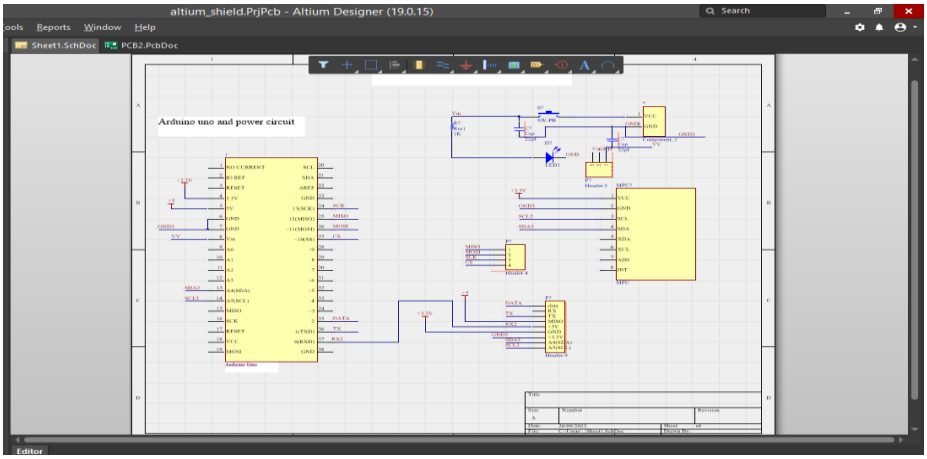

Figure 49_PCB1 Sheet

1.6 PCB1 :

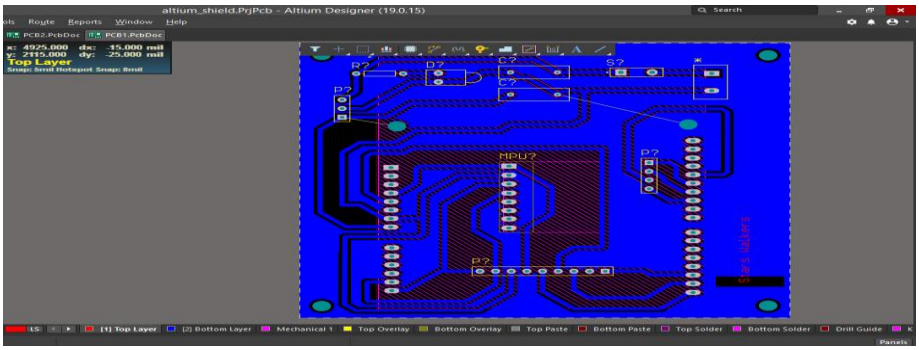

Figure 50_PCB 1

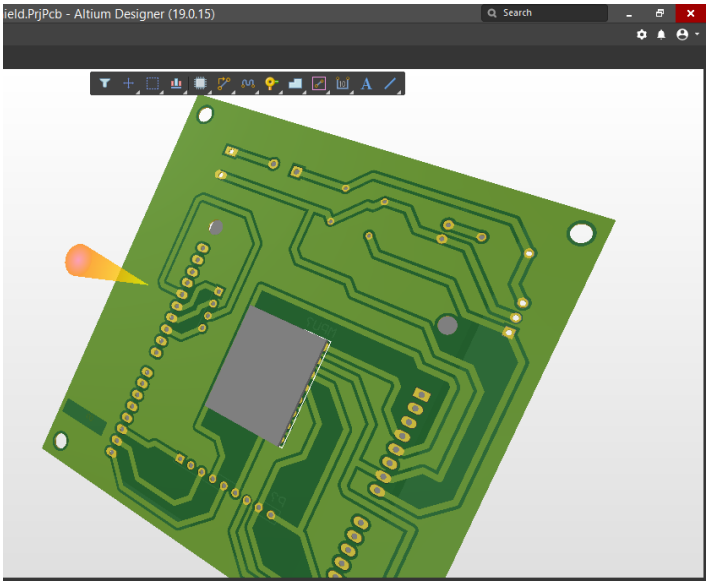

Figure 51_PCB1 3D

---

2) Second PCB :

- it contains SENSORS : DHT11 , BMP180 , GPS NEO6M , SD , RF and male headers

Component	Hole size	Width of pads	Tracks size
Dualling-line packages(DIL	1mm	1.5mm	1mm

- 1.1 DHT11 : to measure temperature and humidity

	VCC	DATA	GND
ARDUINO	5V	PIN (2) no.35 in Arduino	
HEADER	PIN 6	PIN 8	

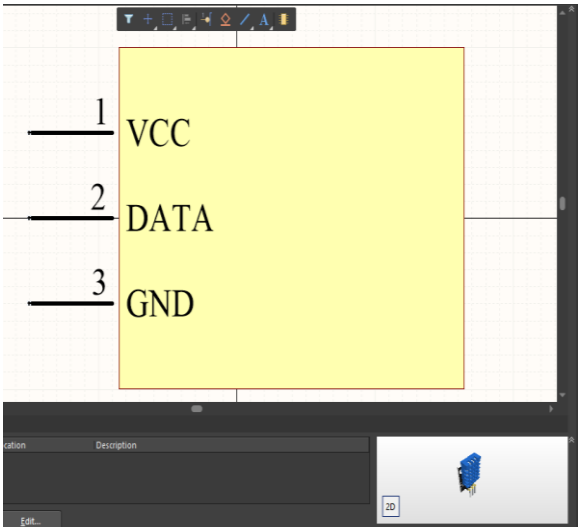

Figure 52_DHT11 SCH-Lib

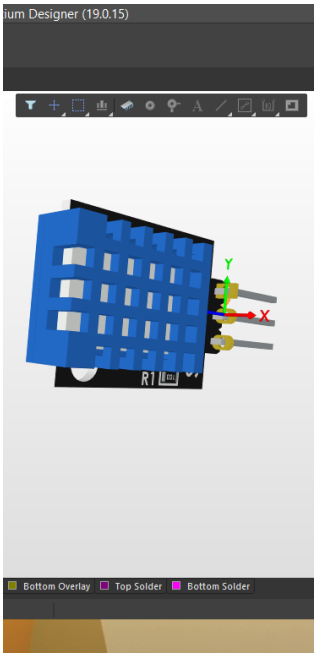

Figure 53_DH11 PCB-Lib

- 1.2 BMP180 : to measure pressure , altitude and temperature .

	VCC	GND	SCL	SDA
ARDUINO	3.3V		A5	A4
HEADER	PIN 3		PIN 2	PIN 1

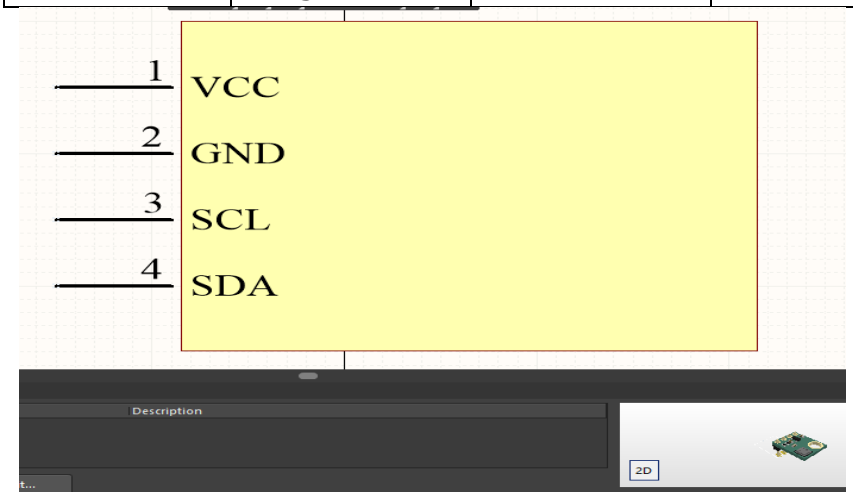

Figure 54_BMP180

- 1.3 GPS NEO 6M : to know the location of the satellite

	GND	TX	VCC
ARDUINO		To RX in Arduino	3.3v
HEADERS		PIN 7	PIN 3

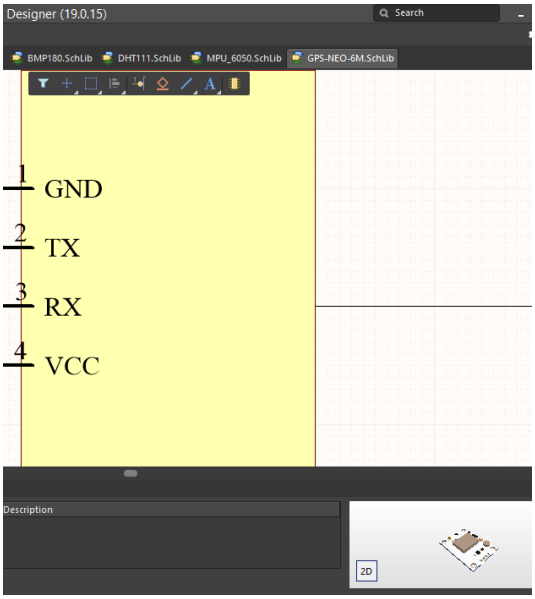

Figure 55_GPS SCH-Lib

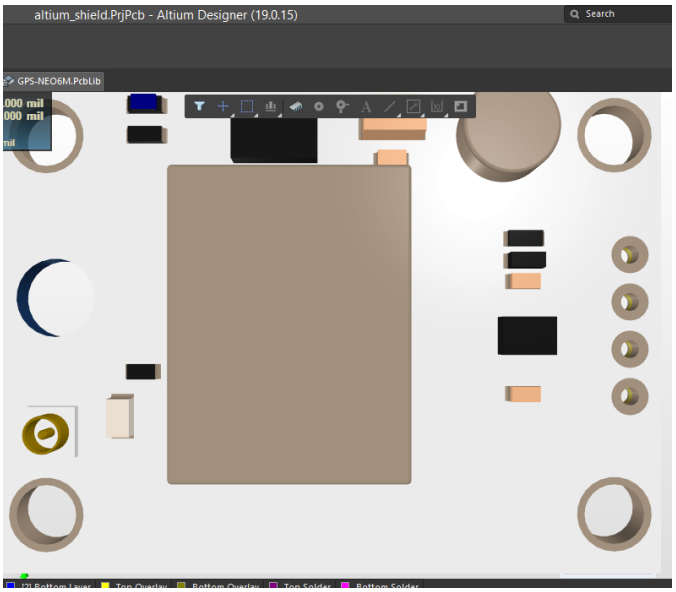

Figure 56_GPS PCB-Lib

- 1.4 Micro SD card : to save data

	GND	VCC	MISO	MOSI	SCK	CS
ARDUINO		5V	PIN 12	PIN ~11	PIN 13	PIN ~10
HEADER		PIN 6	PIN 9	PIN 10	PIN 11	PIN 12

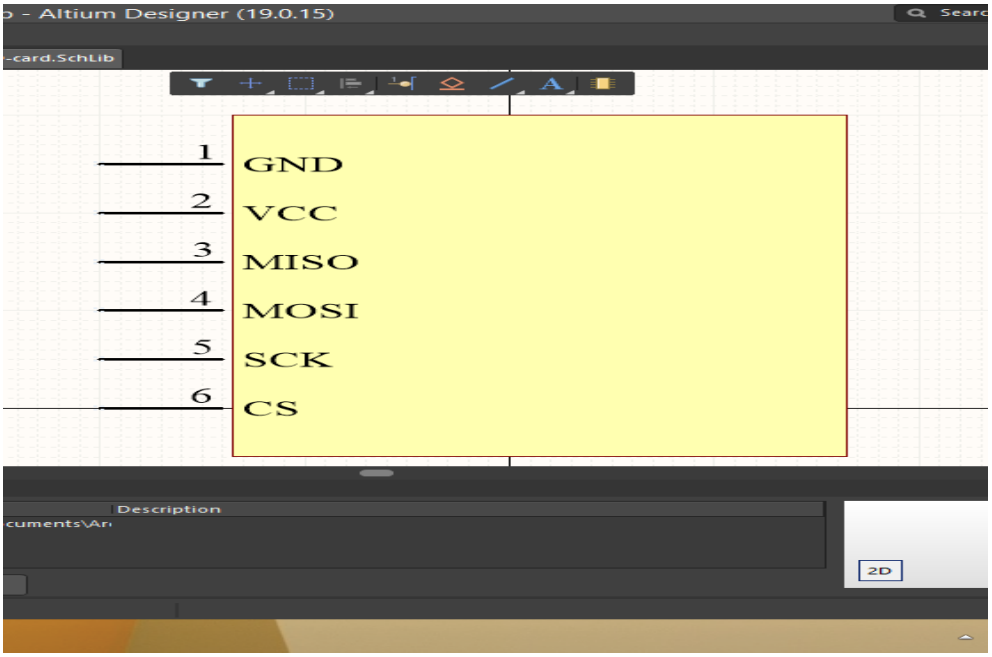

Figure 57_SD SCH-Lib

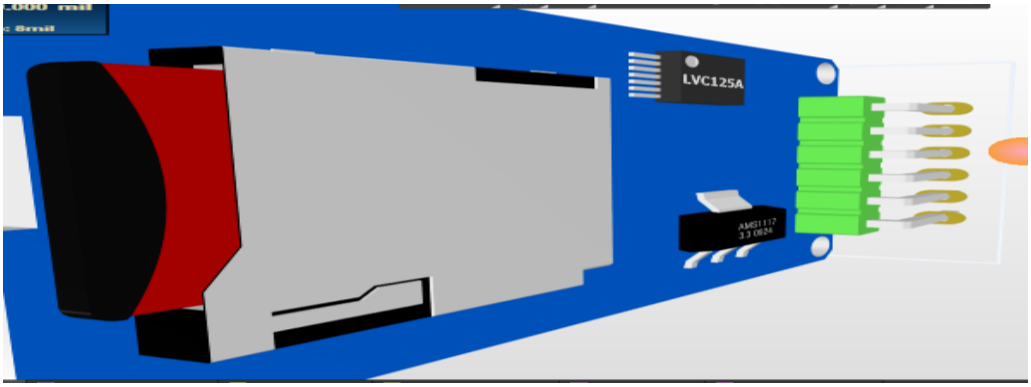

Figure 58_SD PCB-Lib

- 1.5 Headers :

Pin 1	2	3	4	5	6	7	8	9	10	11	12
SCL	SDA	GND	3,3V	RX	5V	TX	DATA	MISO	MOSI	SCK	CS

male headers .

-link between PCB boards .

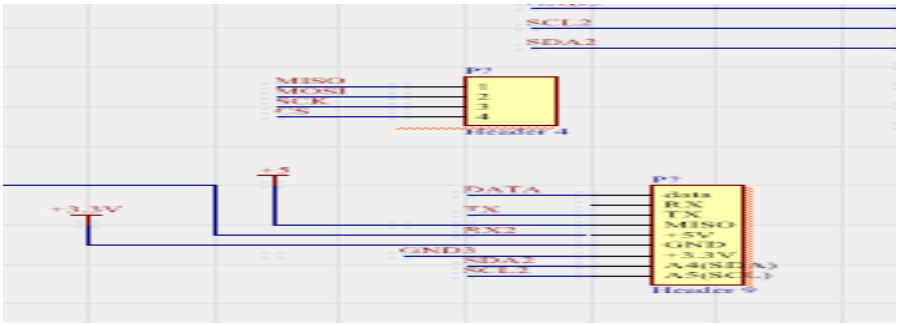

Figure 59_Headers

- 1.6 RF : to transmitter

RX	GND	VCC
TO TX in Arduino		5V
PIN 7 IN Header		PIN 6 IN Header

Figure 60_RF SCH-Lib

- SCHEMATICS :

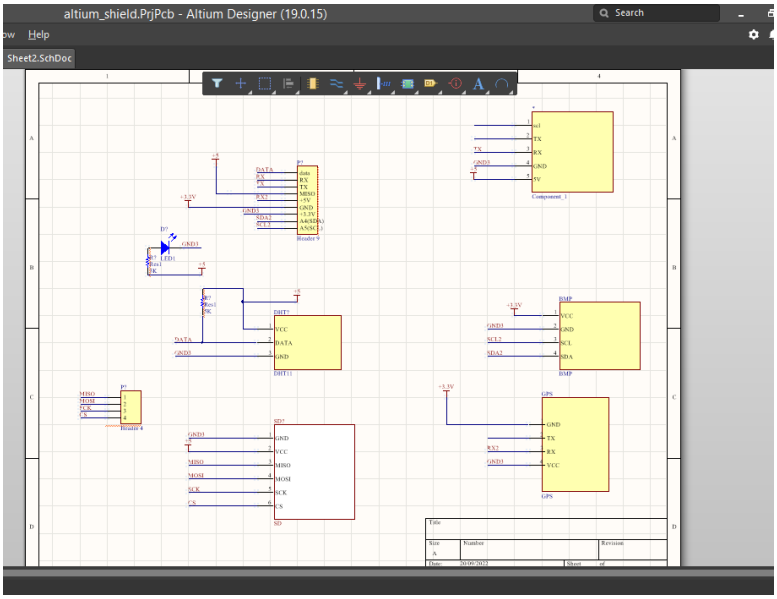

Figure 61_PCB2 Sheet

- PCB2 :

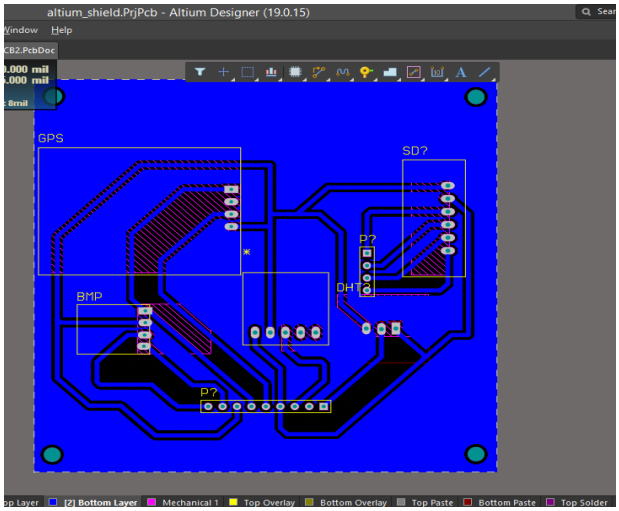

Figure 62_PCB2

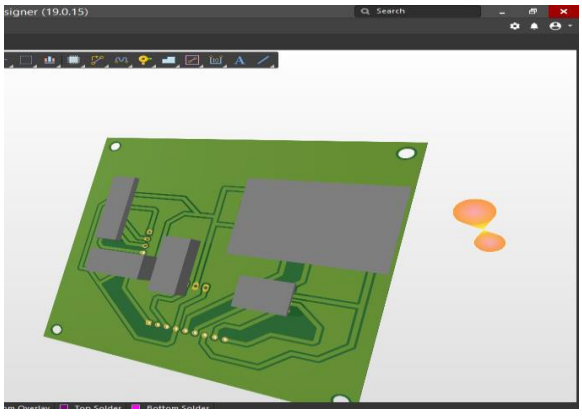

Figure 63_PCB2 3D

Development of CubeSat Ground-Station using LabVIEW

This section discusses the development of CubeSat's ground-station software using LabVIEW. The early development of CubeSat used a freeware 'Serial Monitor' program for the ground-station data logging application, which could only display the received data string, making it difficult to visualize and observe how the sensors' values change in real-time during the descent of CubeSat. Thus, a new software tool was proposed for ground-station to visualize real-time sensors' data in graphical and tabular manner in an intelligible format, and to log the data in the ground-station using LabVIEW.

The aim of the new software tool at the ground station is to give real-time visualization of sensors' data in graphical and tabular manner in human readable format, and log those data in the ground-station computer

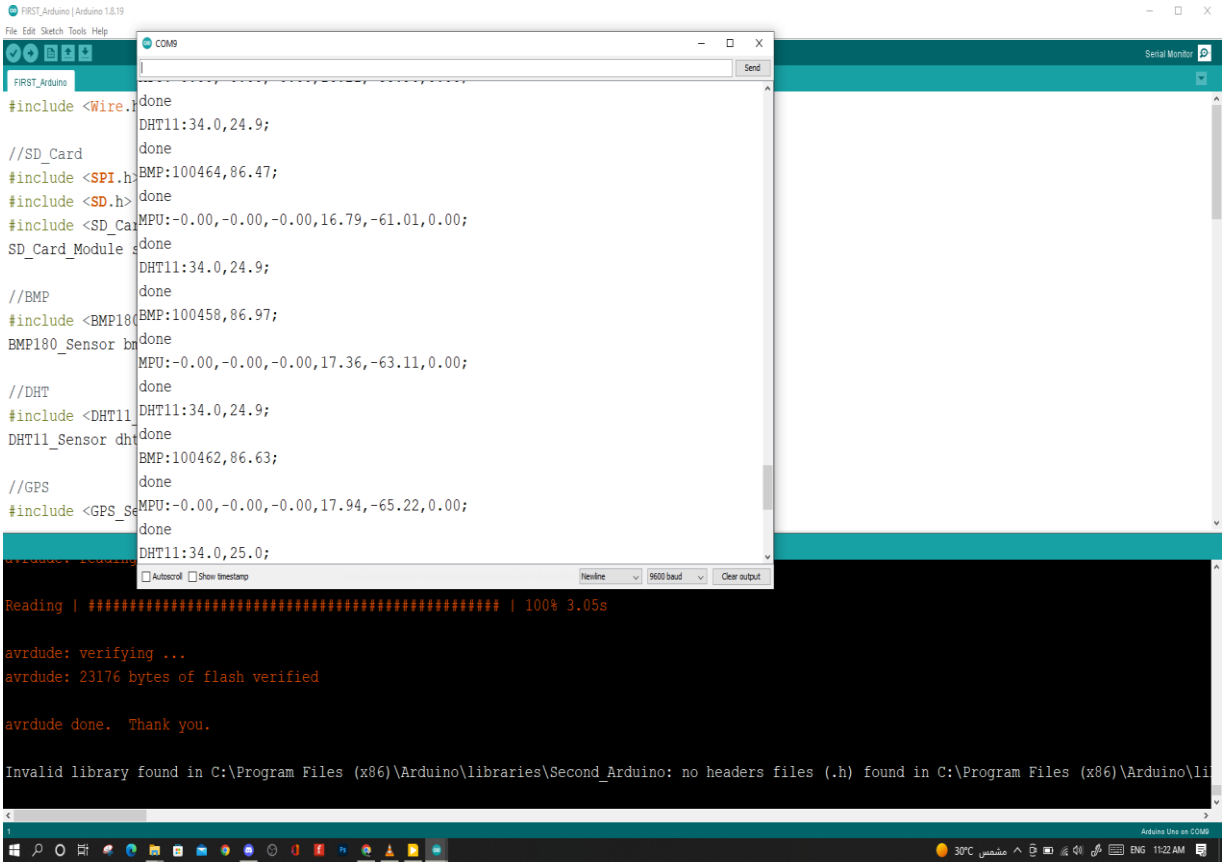

Figure 64_Raw data using serial monitor in Arduino IDE

GROUND-STATION HARDWARE:

The CubeSat's ground-station consists of a computer and an RF communication system. The computer is used for executing ground-station software, and data-logging. RF communication and its interface with the ground-computer, consists of the following components:

Components	Function	No. of components
ATmega16U2	Micro controller	X1
CC1101 chip	Work as RF-Module	X1
Debug cable	Power Arduino	X1

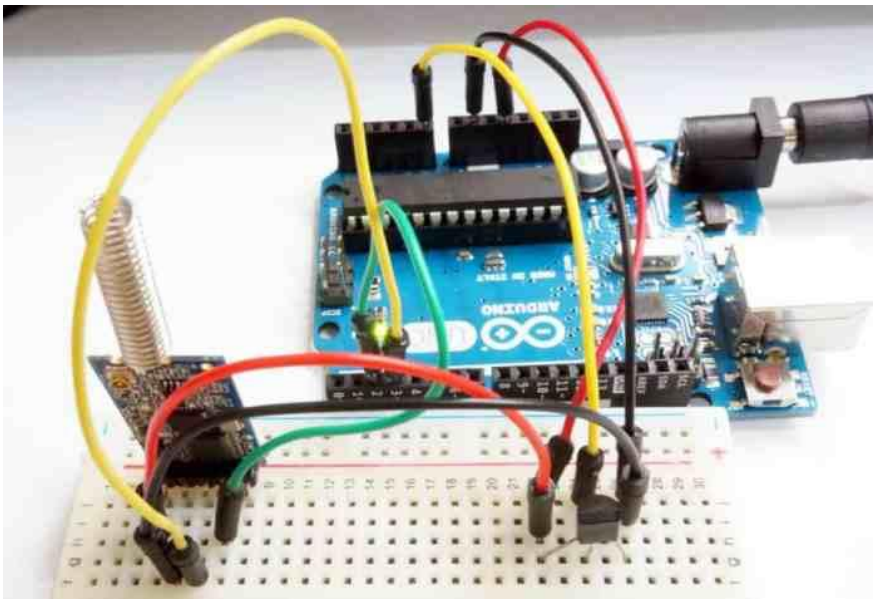

Figure 65_RF communication system to receive data from the Cubesat

GROUND-STATION SOFTWARE:

The main purpose of Ground-Station software is to visualize and observe how the sensors' values are changing in real-time, during descend of CubeSat. Previously, serial monitor was used to display the received data. But it was difficult to visualize and observe the sensors' values from the string of data.

Thus, a new software tool for ground-station is developed. The aim of the user interface of the ground station is to give real-time visualization of sensors' data in graphical and tabular manner in human readable format, and log those data in the ground-station computer.

Figure 66_Graphical user interface

Figure 67_Graphical user interface

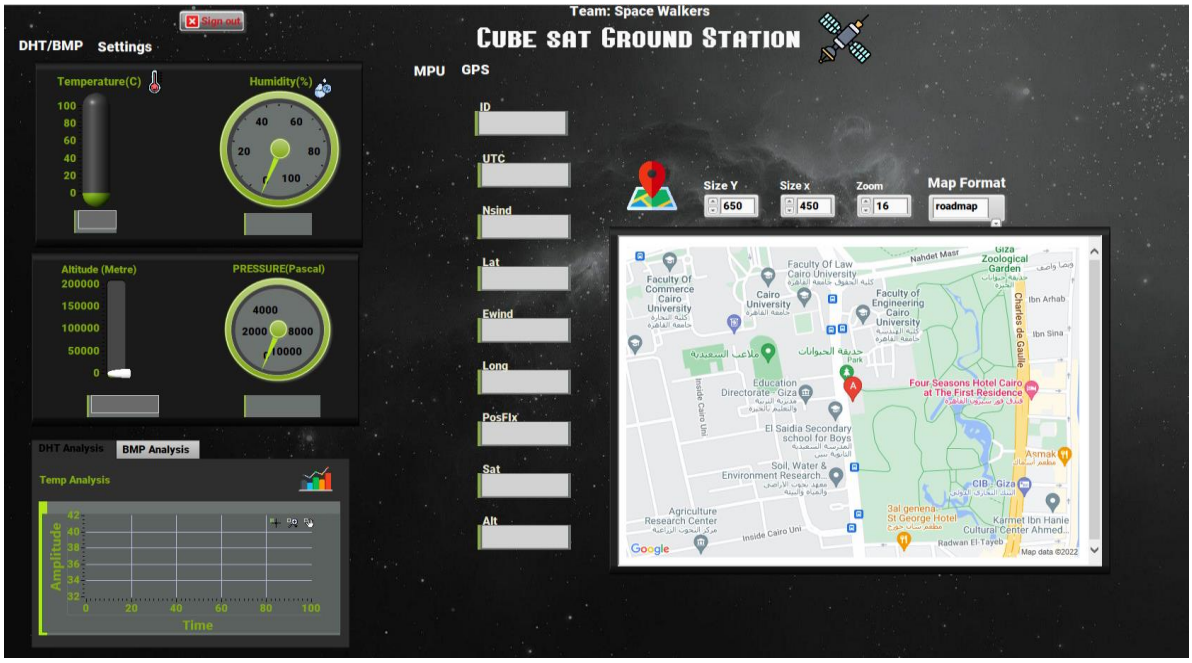

Figure 68_Graphical user interface

Figure 3 shows the screenshot of the UI of software that was developed in LabVIEW. The user needs to specify the COM Port address where the RF receiver is connected.

Figure 69_Arduino Interface

The user needs to specify the COM Port address as in **Figure 69**, after connecting to the RF receiver and starting the application, the software displays the data send by CubeSAT. Furthermore, user can see the plot of different variable with respect to the time and can also see the tentative rotation of CubeSAT in 3D Model.

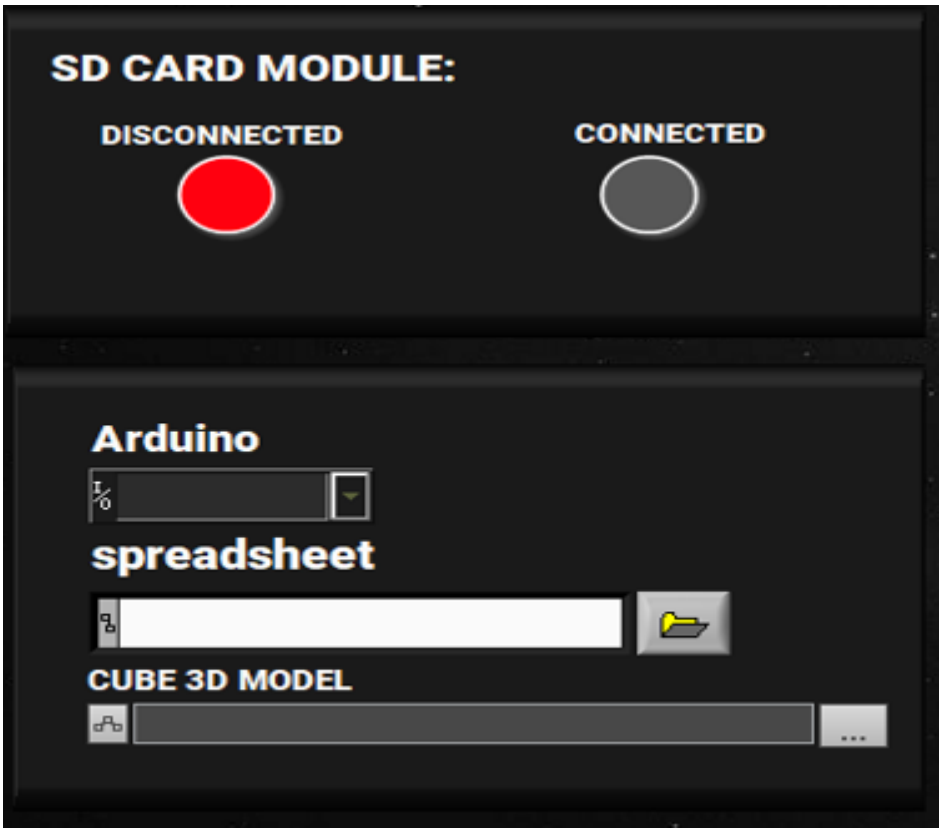

Figure 70_Saving received data

Also, user can log the data to excel file in ground station using spreadsheet button as in **Figure 70**.

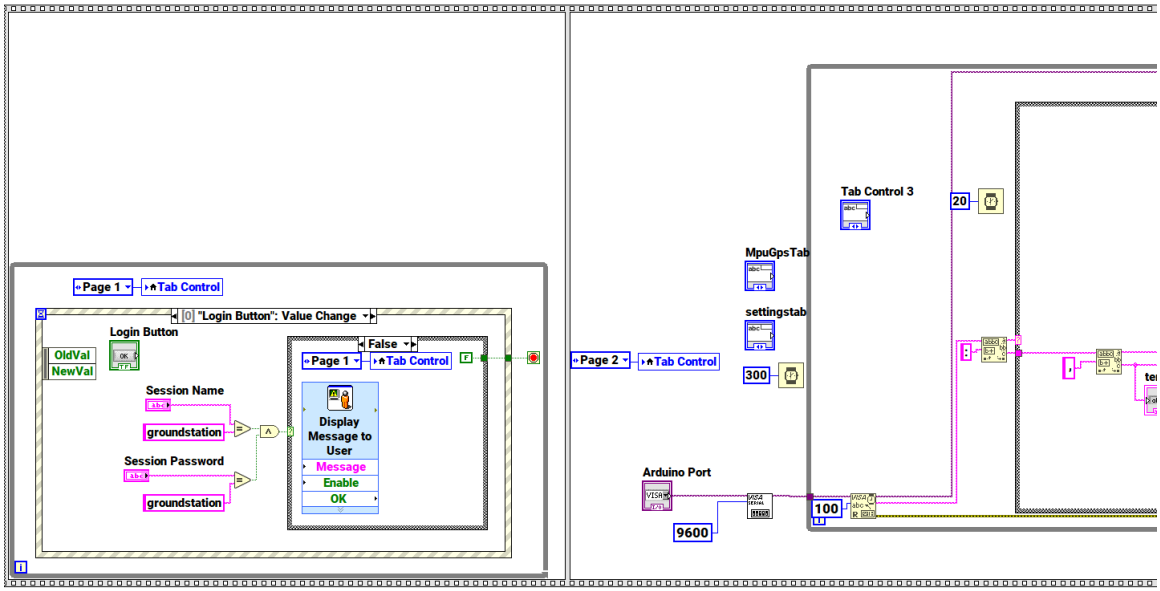

Figure 71_Block Diagram of Ground Station Software in LabVIEW

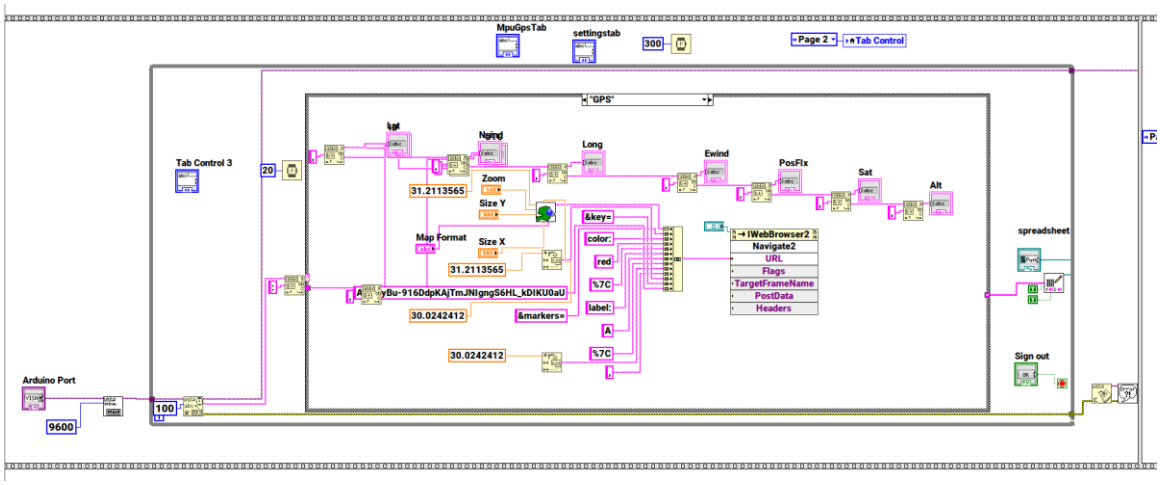

Figure 72_Block Diagram of Ground Station Software in LabVIEW (GPS Sensor)

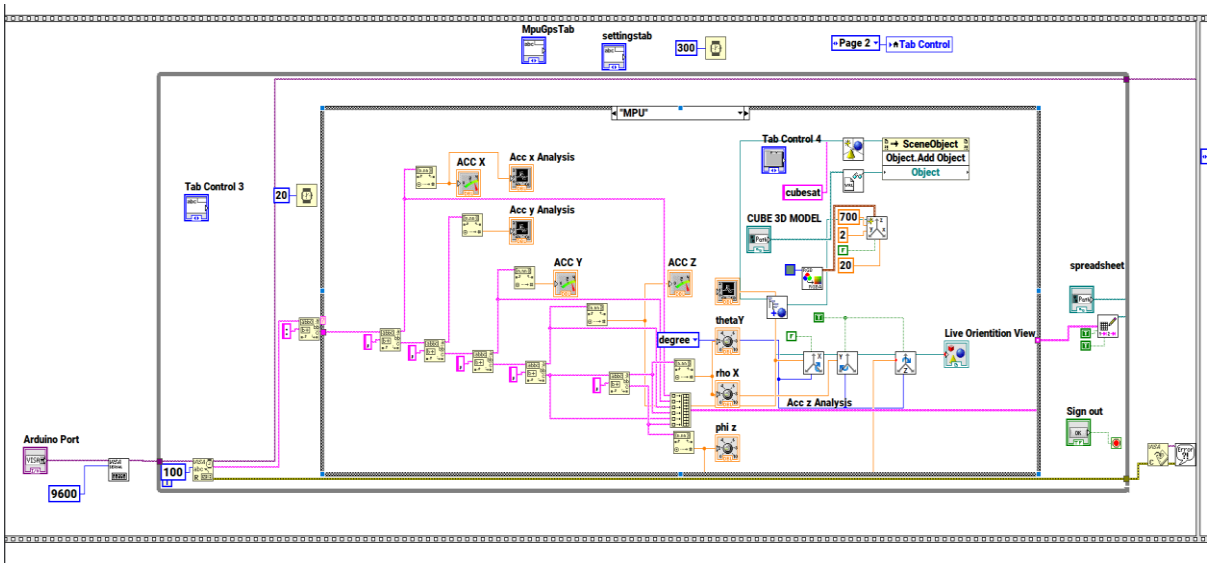

Figure 73_Block Diagram of Ground Station Software in LabVIEW (MPU-6050 Sensor)

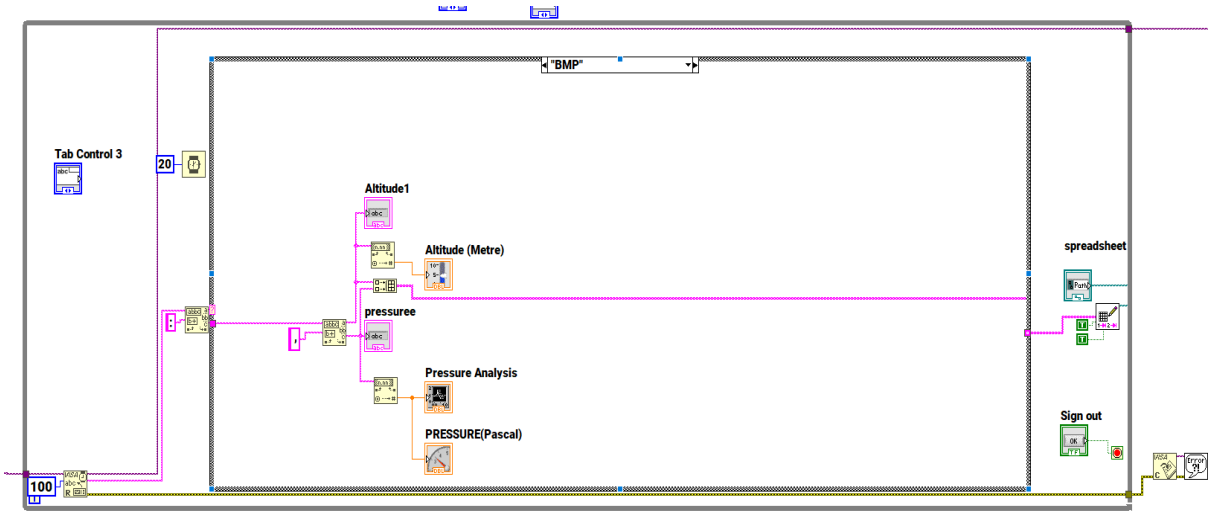

Figure 74_Block Diagram of Ground Station Software in LabVIEW (BMP 180 Sensor)

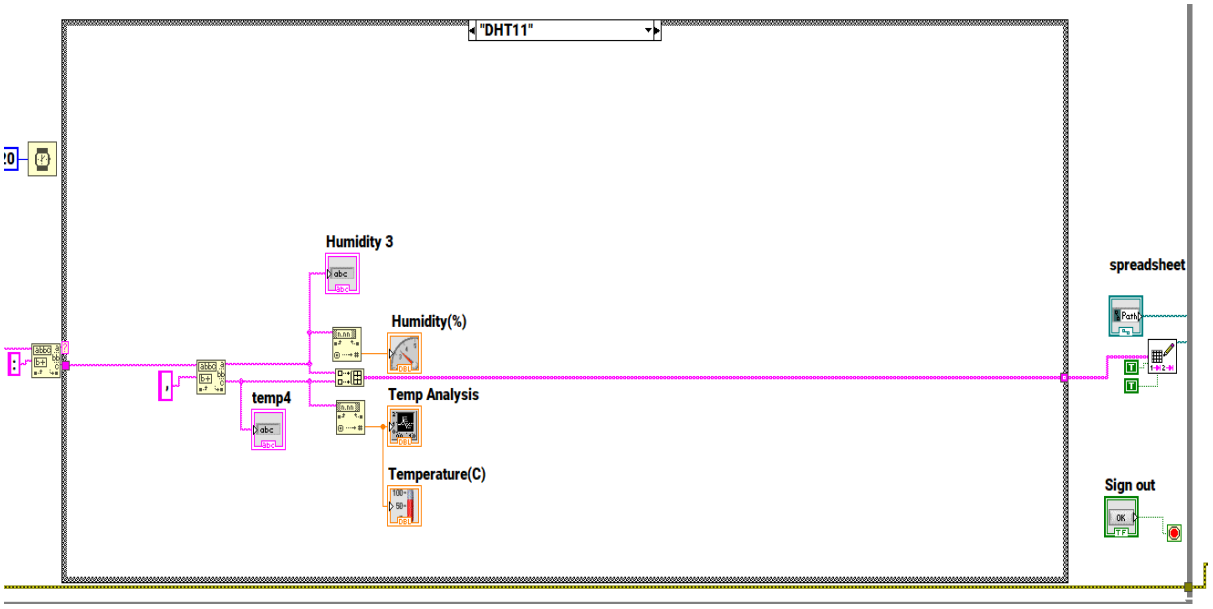

Figure 75_Block Diagram of Ground Station Software in LabVIEW (DHT 11 Sensor)

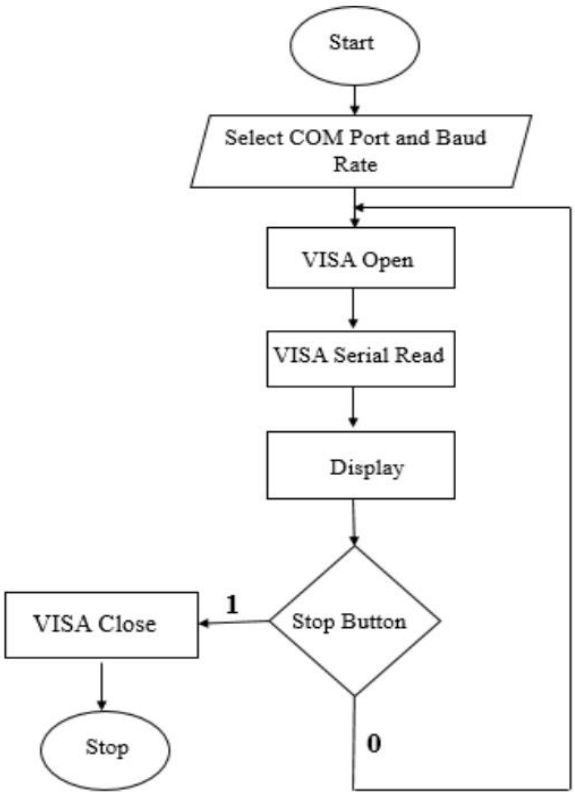

Figure 76_Flow chart diagram for the program

Figure 76 shows the flowchart of program flow in LabVIEW. At first the Communication port and Baud Rate needs to be selected to communicate with CanSat. Visa is opened after configuring the COM Port and Baud Rate. Serial Read block reads the data available in the serial buffer which is then displayed to the user. Whenever the sign out button is pressed the program is terminated

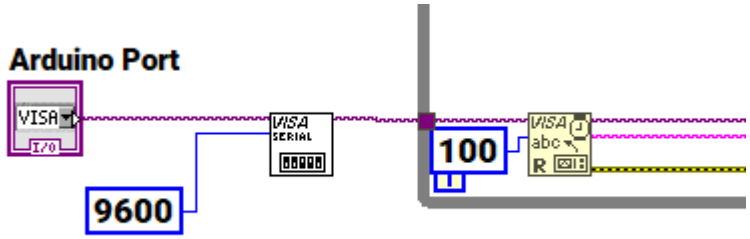

Figure 77_Visa configure port

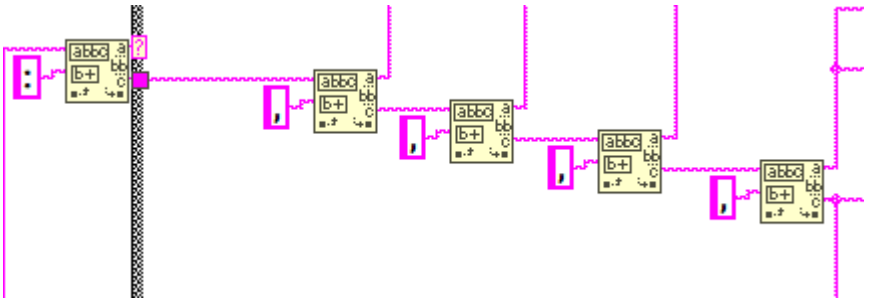

Figure 78_Match pattern used to separate sensors' data

The data rate and COM Ports are configured using VISA Configure Serial Port Block. The configured output is then fed in to Visa Read block which reads the data as a string and stores as a string constant. The string which contains the value of sensors is then separated by match pattern

and stored as an array of strings. Each element of array contains the value of each sensor. The elements of array are then converted as an integer and then displayed to the user until the stop button is pressed. Data transmitted by RF transmitter, which consists of sensors values separated. For data logging,

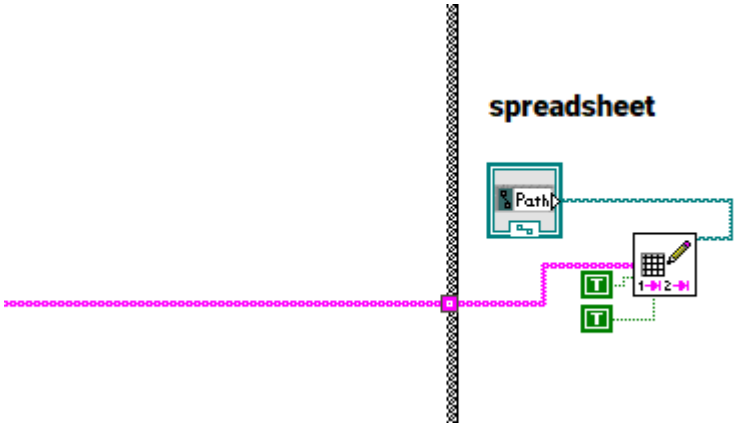

Figure 79_Saving received data in excel file

The received string is converted to array using Spreadsheet string to array block. The output array is then delaminated by the comma (,) character so that it can be written in each cell in Excel. The output data in excel file is saved in the desktop within chosen folder.

Future Work (Using Solar Cells and a Solar Tracking System for the Electric Power Subsystem)

The idea is based on a dual-axis solar tracker controlled with Arduino Uno. The solar tracker can be controlled automatically with the help of Light Dependent Resistor (LDR) sensors or manually using a potentiometer. Moreover, this test bench provides virtual instrumentation based on Excel in which its solar tracker data can be recorded and presented.

It is based on a solar tracker that can rotate automatically to track the sun with the help of four LDR sensors and two servomotors (SM1 and SM2), or manually using a potentiometer. To switch between the two modes (automatic and manual), a push-button is used. Another push-button is used to link either the SM1(up-down servomotor) or SM2 (left-right servomotor) to the potentiometer to control their movement. Moreover, a computer is used as a virtual instrument to visualize the mode and current, voltage and power of the PV panel according to time in MS Excel. Arduino Uno board is utilized to implement all software requirements of the system.

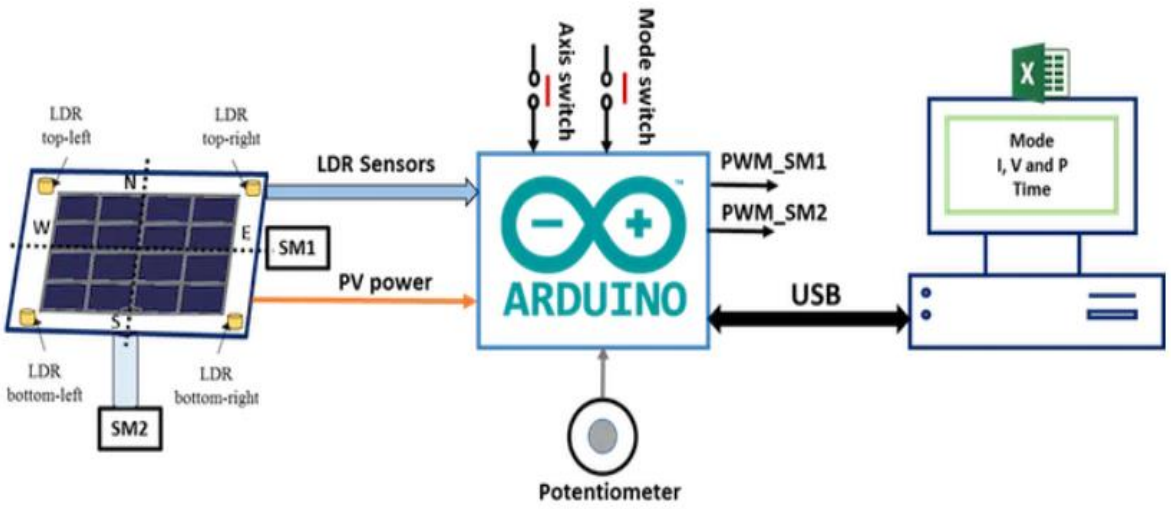

Figure 80_Solar tracker system diagram

The Solar panels to be used are 1 Watt - 5v/200mA. This is a custom solar panel that has a high efficiency at 15.5%.

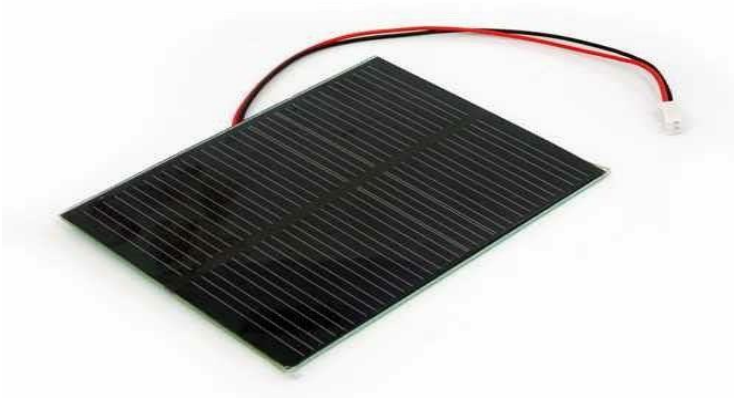

Figure 81_Solar Panel

Abstract

Our project is a CubeSat controlled by the Arduino microcontroller, with an internal measurement unit to measure the acceleration and gyroscopic angles, a temperature and humidity sensor and a sensor to measure the pressure and the altitude. The data was received at a ground station designed by LabView.

Team Formation

Coding and Control /	Ahmed Abd El-Kareem 421918
PCBs Design /	(Jannat-Allah Sabry 232029) – (Seif Atef 242123)
Structural Design and Documentation /	Ahmed Khalil 232125
Ground Station /	Kareem Elfaqy 211832

List Of Figures

References

https://lastminuteengineers.com/mpu6050-accel-gyro-arduino-tutorial/

https://dronebotworkshop.com/mpu-6050-level/

https://www.adafruit.com/product/386

https://lastminuteengineers.com/bmp180-arduino-tutorial/

https://lastminuteengineers.com/neo6m-gps-arduino-tutorial/

https://lastminuteengineers.com/arduino-micro-sd-card-module-tutorial/https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/

https://www.youtube.com/watch?v=bj-fc7qO0Wc

https://www.youtube.com/watch?v=jPHsPpTEt6A

https://www.youtube.com/watch?v=oak4UTERmCk

https://www.youtube.com/watch?v=OCV27Gwm88Q

https://youtu.be/rCf4sd_YXnA