Rocket journey

HRIDOY RANJAN KALITA

August 2021

•

1 Introduction

Designing a model rocket is complicated in many different ways,but it is more challenging to figure it out the mistakes.At every step one need to understand what is going on and what may be the output.

"Experience is a hard teacher because she gives the test first ,the lesson afterward" Verson Law.

From the above line we need to understand that there might be lots of information ,but still the main purpose of this whole project is to give you a basic introduction about model rocket ,so that one can take help from here and can make his own rocket and learn from errors and experience.

-

2 Material used

Components:
1.Arduino NANO or similar clone (should use an ATMEGA 328 chip or Uno for same results)*
2.Inertial measurement unit - I used the MPU 6050
3.temperature and pressure sensor - I used the BMP 180,
4.microSD card reader with SPI bus microSD card with SD adapter
(5) 5mm LEDs - I used a red, yellow, and blue one normally open pushbutton or tactile switch
(6) 220 ohm resistors (or similar value) 5k ohm resistor (or similar value)
7.piezo buzzer or small speaker 2-6 additional header pins (optional) several lengths of 22 awg wire (or smaller)
8.9 volt battery connector or other power adapter

1. Materials:

through-hole PCB for prototyping (I used 50 mm x 70 mm perfboard) 3D printed enclosure or plastic plate (optional - useful for protecting the flight computer during flight), Balsa wood,PVC tube ,Rocket Motors, Ignitors,CardBoard,

•

3 What constitute a model rocket?

Nose cone

The nose cone of the rocket has a shape that causes the air to flow smoothly around the rocket. It could be conical in shape, but at subsonic speeds a rounded shape gives lower aerodynamic drag. The nose cone is typically made from plastic, balsa wood, hardwood, fiberglass, or styrofoam. It can be either hollow or solid.

Payload Section

Not all rockets have a payload section. The model shown has a clear plastic payload section that allows any payload inside to be easily inspected visually. The payload section can be used to carry a variety of payloads, such as electronic altimeters or cameras.

Transition Section Transition Section A transition section is used to connect body tubes of different diameters. Not all rocket designs incorporate a transition. The transition could be used to either increase or decrease the rocket's diameter at that point. Transition sections are typically made from plastic, balsa wood, hardwood, fiberglass, or paper. They may be either hollow or solid. In the model shown, the bottom of the transition is where the rocket separates when the parachute is elected.

Shock Cord Mount

Shock Cord Mount The shock cord must be attached to the body of the rocket. There are many ways to do this, but the most common used in model rockets is a folded-paper mount glued to the inside of the body tube. It is also common to connect the shock cord (or a separate anchor line) to the front of the motor mount in larger-diameter rockets.

Shock Cord Shock Cord The shock cord holds the parts of the rocket together after they separate at ejection. The shock cord may be made of an elastic material to help absorb the shock of the separating parts coming to a halt at the ends of the cord, or it could be made from a non-elastic line (in which case it is normally longer). Typical materials for shock cards are sewing elastic, rubber, nylon, and Kevlar.

Parachute

All model rockets require a recovery system to slow their descent and return them safely to the ground. The most common type of recovery system is the parachute. The parachute may be made from thin plastic or cloth. The parachute is expelled from the body tube by the ejection charge of the rocket motor after a delay to allow the rocket to reach apogee and be traveling at a relatively slow speed. Other recovery systems include streamer, featherweight, glide, helicopter, body drag, and tumble.

Shroud lines

The shroud lines connect the parachute canopy to the rest of the rocket. The shroud lines on most model rocket parachutes are made of strong thread, such as carpet thread, but they may also be made of other material. The number of shroud lines varies, but is typically 6 or 8 lines on a model rocket parachute. More shroud lines can cause a simple flat parachute (a "parasheet") to form into a more nearly spherical shape, and therefore be more efficient.

Recovery Wadding

Recovery wadding is flame-resistant material that protects the parachute (or other recovery system components) from the hot blast of the motor ejection charge. The ejection charge would melt a plastic parachute, so this protections is necessary. Recovery wadding is typically chemically treated tissue paper or cellulose insulation. It is vital that only flame-resistant materials be used as recovery wadding to prevent the ejected wadding from causing fires.

Body Tube

The body tube (or tubes) are the airframe of the model rocket. Body tubes are typically made from paper, fiberglass, or plastic, with the spiral-wound paper tube being the most common. The rocket may have multiple body sections connected with transition sections (if the tubes are different diameters) or nose blocks or couplers (if the tubes are the same diameter). The body tube usually contains an engine mount to hold the motor, and space for the recovery system.

Launch Lug

When a model rocket first begins to lift off, it is traveling too slowly for the fins to provide aerodynamic guidance, so the rocket must be guided for the first few feet by a launch rod or rail. The launch lug is what allows the model rocket to slide along the rod. On a model rocket, the launch lug is typically a small diameter tube. Larger rockets may use rail buttons on the side of the rocket to allow it to slide along a much stiffer launch rail for initial guidance.

Fins

The fins of the rocket provide aerodynamic stability in flight so that the rocket will fly straight (in the same way that the feathers of an arrow help it fly straight). The fins are typically made from plastic, balsa wood, plywood, cardboard, or fiberglass. A rocket three or four fins, but may have more. Some rockets don't have any fins and may rely upon a cone or other surfaces to stabilize the model in flight. On larger rockets, the fins may be mounted through slots in the body tube for extra strength.

Engine Block

The engine block, or thrust ring, keeps the rocket motor from moving forward into the rocket body during the thrusting phase of the flight. Engine blocks are typically thick paper rings that are glued into the motor mount tube. If the rocket body has a larger diameter than the motor, the motor mount tube that holds the rocket motor will be centered within the body tube using cardboard or plywood centering rings.

Rocket Engine

The engine, or motor, of the model rocket is a commercially manufactured solid-propellant rocket motor that is good for one flight. Model rocket motors are typically made from thick wound paper tubes. The motor contains a ceramic nozzle, a solid propellant grain (chemically similar to black powder, but compressed into a solid piece), a slow-burning delay element, and a loose-grained ejection charge that is retained by a clay cap. Larger rockets may use motors with plastic casings and ammonium perchlorate composite propellant. Some motors use metal casings that can be reloaded with commercially manufactured APCP grains.

Igniter

Model rocket engines are always ignited electrically from a safe distance. The igniter (which is sold with the motor) is typically made from wires that connect to a thin wire coated in pyrogen. This pyrogen-coated tip is inserted into the rocket motor's nozzle and in contact with the solid propellant. When sufficient electrical current is passed through the igniter, the thin wire heats, igniting the pyrogen, which then ignites the motor propellant.

•

4 Different parts of a rocket

1. **control system**

One of the most important part is the part that control the rocket,specially the rockets which is govern by the thrust vector system.But here we aren't going to use this as this type of rockets need very high level of precision and experience.Hence thrust vector control rocket is out of this scope and a future project.

But here we are going to make the rocket a parachute deployment one, hence we need a bit of control over the rockets.

1. .1 Arduino

Arduino is one of the most favourite microcontroller among people,specially Arduino UNO and Nano. It is the brain of the rocket,allows different sensors to work properly.

1. .2 BMP180

BMP180 is a very cheap sensor ,used for measuring pressure , altitude,temperature etc.In this project we will use it for the parachute deployment purpose. Aim is that this sensor will calculate the height so that at a certain height we can deploy out parachute. But before directly use it in our project it is always benifecial for us to check about its performance.

The connections are shown bellow on figure 3

Now let us look at the following sensor data in figure 4:

For the above test the code used is given below:

Figure 1: Arduion Uno

```
#include <SFE\_BMP180.h>
#include <Wire.h>

// You will need to create an SFEBMP180 object, here called "pressure":

SFE\_BMP180 pressure;

#define ALTITUDE 1655.0 // Altitude of SparkFun's HQ in Boulder, CO. in
    meters


void setup()
{
  Serial.begin(9600);
  Serial.println("REBOOT");

  // Initialize the sensor (it is important to get calibration values
    stored on the device).

  if (pressure.begin())
    Serial.println("BMP180 init success");
  else\\
  {
    // Oops, something went wrong, this is usually a connection problem,\\
    // see the comments at the top of this sketch for the proper
    connections.\\

    Serial.println("BMP180 init fail\n\n");
    while(1); // Pause forever.
```

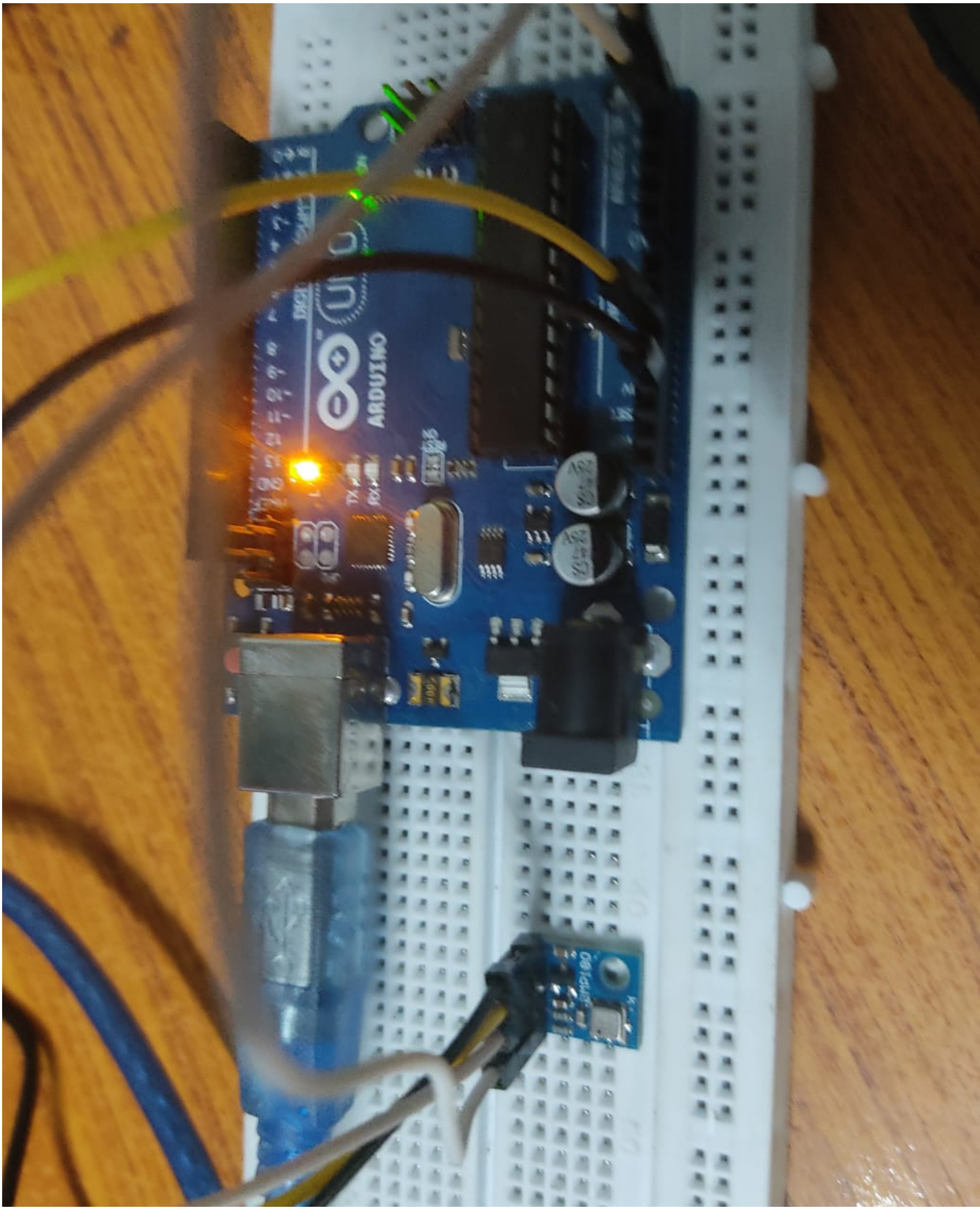


Figure 2: Pressure sensor

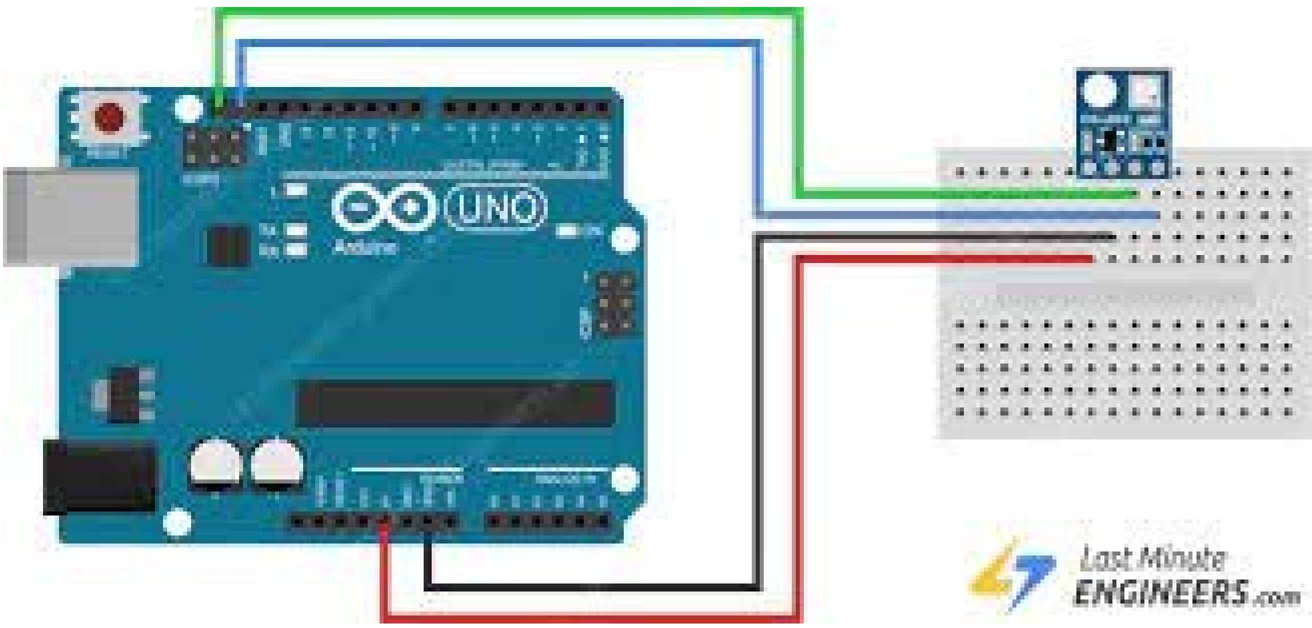


Figure 3: sensor connections

```
  }
}


void loop()
{
  char status;
  double T,P,p0,a;

  // Loop here getting pressure readings every 10 seconds.

  // If you want sea-level-compensated pressure, as used in weather
    reports,

  // you will need to know the altitude at which your measurements are
    taken.
  // We're using a constant called ALTITUDE in this sketch:

  Serial.println();
  Serial.print("provided altitude: ");
  Serial.print(ALTITUDE,0);
  Serial.print(" meters, ");
  Serial.print(ALTITUDE*3.28084,0);
  Serial.println(" feet");

  // If you want to measure altitude, and not pressure, you will instead
    need
  // to provide a known baseline pressure. This is shown at the end of the
    sketch.
```

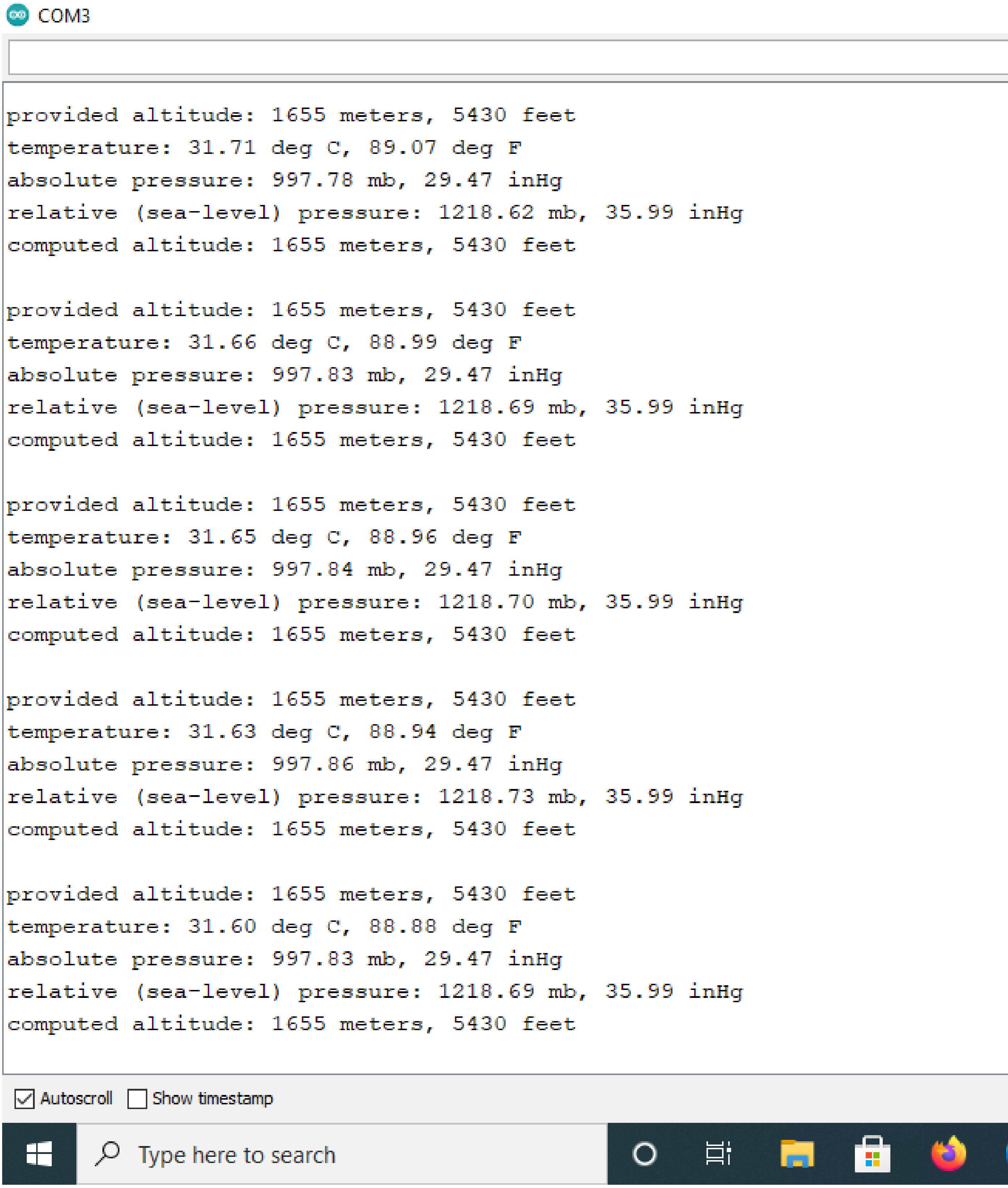


Figure 4: different parameters that can be measured

```
// You must first get a temperature measurement to perform a pressure reading.

// Start a temperature measurement:
// If request is successful, the number of ms to wait is returned.
// If request is unsuccessful, 0 is returned.

status = pressure.startTemperature();
if (status != 0)
{
  // Wait for the measurement to complete:
  delay(status);

  // Retrieve the completed temperature measurement:
  // Note that the measurement is stored in the variable T.
  // Function returns 1 if successful, 0 if failure.

  status = pressure.getTemperature(T);
  if (status != 0)
  {
    // Print out the measurement:
    Serial.print("temperature: ");
    Serial.print(T,2);\\
    Serial.print(" deg C, ");\\
    Serial.print((9.0/5.0)*T+32.0,2);
    Serial.println(" deg F");

    // Start a pressure measurement:
    // The parameter is the oversampling setting, from 0 to 3 (highest res, longest wait).
    // If request is successful, the number of ms to wait is returned.
    // If request is unsuccessful, 0 is returned

    status = pressure.startPressure(3);
    if (status != 0)\\
    {
      // Wait for the measurement to complete:
      delay(status);

      // Retrieve the completed pressure measurement:
      // Note that the measurement is stored in the variable P.
      // Note also that the function requires the previous temperature measurement (T).
      // (If temperature is stable, you can do one temperature measurement for a number of pressure measurements.)
      // Function returns 1 if successful, 0 if failure.

      status = pressure.getPressure(P,T);
      if (status != 0)
      {
        // Print out the measurement:
```

```
          Serial.print("absolute pressure: ");
          Serial.print(P,2);
          Serial.print(" mb, ");
          Serial.print(P*0.0295333727,2);
          Serial.println(" inHg");

          // The pressure sensor returns abolute pressure, which varies with altitude.
          // To remove the effects of altitude, use the sealevel function and your current altitude.
          // This number is commonly used in weather reports.
          // Parameters: P = absolute pressure in mb, ALTITUDE = current altitude in m.
          // Result: p0 = sea-level compensated pressure in mb

          p0 = pressure.sealevel(P,ALTITUDE); // we're at 1655 meters (Boulder, CO)
          Serial.print("relative (sea-level) pressure: ");
          Serial.print(p0,2);
          Serial.print(" mb, ");
          Serial.print(p0*0.0295333727,2);
          Serial.println(" inHg");

          // On the other hand, if you want to determine your altitude from the pressure reading,
          // use the altitude function along with a baseline pressure (sea-level or other).
          // Parameters: P = absolute pressure in mb, p0 = baseline pressure in mb.
          // Result: a = altitude in m.

          a = pressure.altitude(P,p0);
          Serial.print("computed altitude: ");
          Serial.print(a,0);
          Serial.print(" meters, ");
          Serial.print(a*3.28084,0);
          Serial.println(" feet");
        }
        else
        Serial.println("error retrieving pressure measurement\n");\\
      }
      else
      Serial.println("error starting pressure measurement\n");
    }
    else
    Serial.println("error retrieving temperature measurement\n");
  }
  else
  Serial.println("error starting temperature measurement\n");

delay(5000); } // Pause for 5 seconds.
```

```
}
```

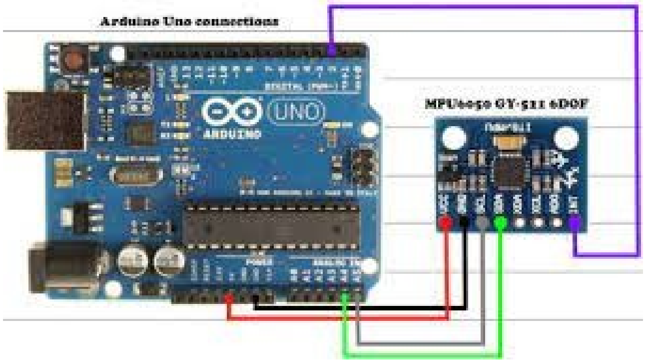


Figure 5: IMUconnection

1. .3 IMU.MPU6050

MPU6050 is also a very useful but cheap sensor used in various applications. One of it's used is calculating roll,pitch ,yaw of a model rocket.It has 6 Degree of freedom.3 axis gyroscope and 3 axis accelarometer with the help of which we can measure the orientations of a rocket.It is the most important part of a thrust vector control rocket.
Testing:
IMU module is not as easy as BMP180,specially while calculating pitch and roll etc. Hence it is always recommended to test before final project.

All the circuit diagram are shown in figure 5.

Now we will calculate the roll and pitch using this sensor.

A simple graph of pitch variation with time is shown in figure7.

From the reading of accelerometer and gyroscoscope we can calculate the roll and pitch .Here the IMU will provide us the acceleration along x,y and z axis(a_x,a_y,a_z respectively).
Now

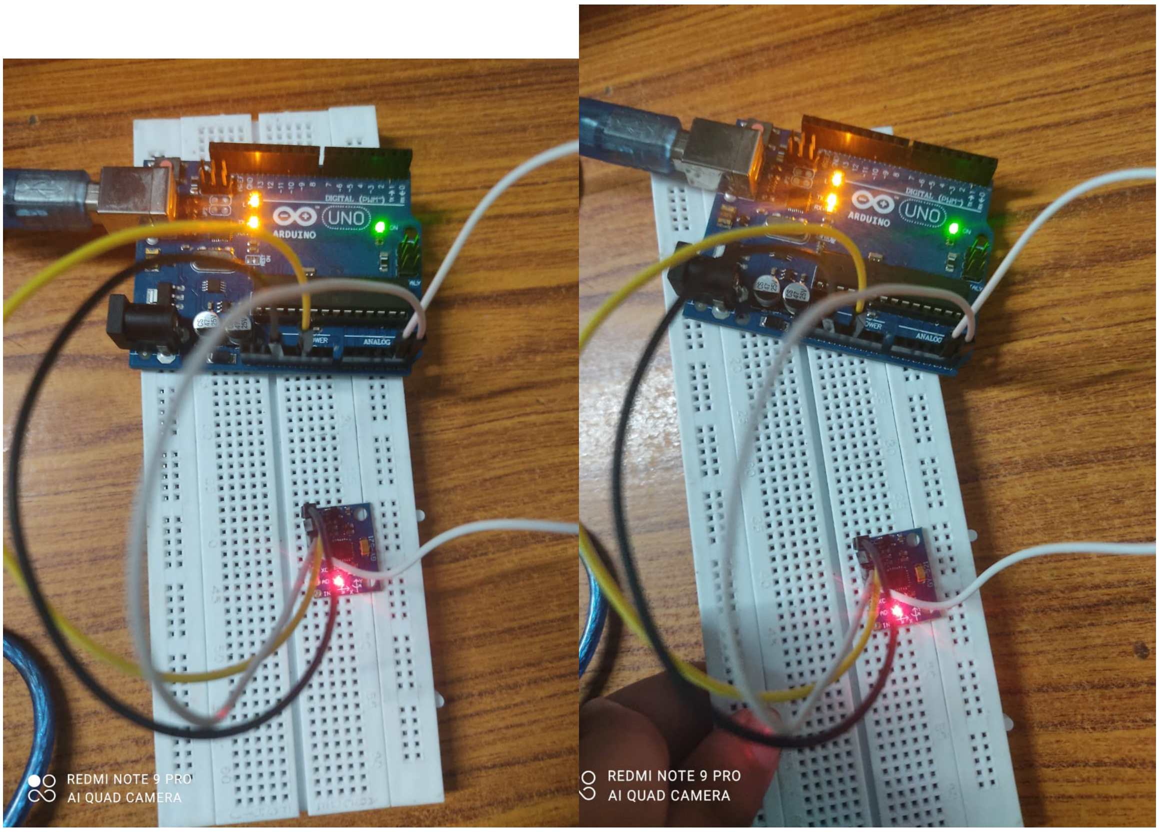
$$p = \arctan(\frac{a\_x}{\sqrt{ay^2 + a\_y^2}}); \phi = \arctan(\frac{a\_y}{\sqrt{ax^2 + a\_y^2}}); \theta \arctan(\frac{\sqrt{ax^2 + a\_y^2}}{a\_z}); \quad (1)$$

At the end we are now going to look at the code of MPU6050 for calculating Pitch angle

```
// Include Wire Library for I2C
#include <Wire.h>

// Define I2C Address - change if reqiuired
const int i2c_addr = 0x3F;

```



(a) board is horizontal to the ground

(b) board makes an angle

Figure 6: Caption

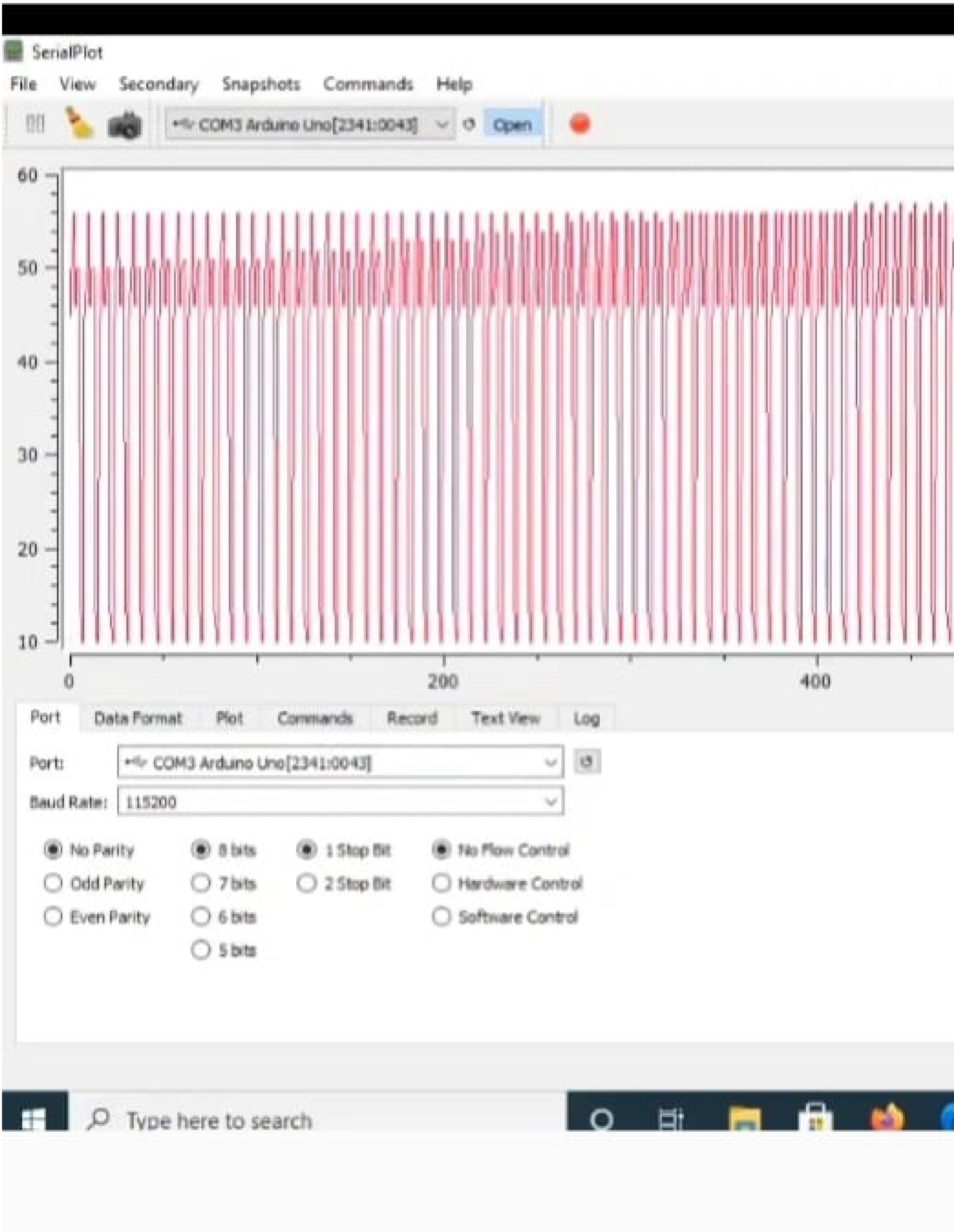


Figure 7: pitch vs time plot

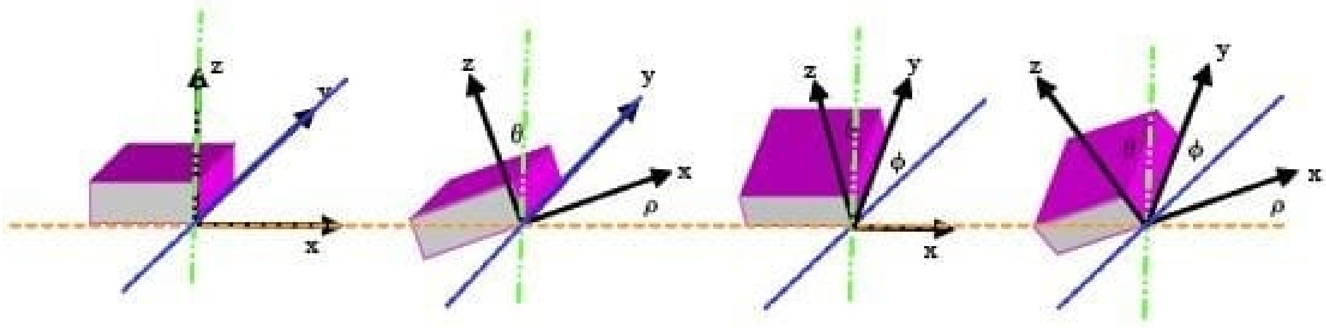


Figure 8. Three Axis for Measuring Tilt

Figure 8: roll,pitch

```
//Variables for Gyroscope
int gyro_x, gyro_y, gyro_z;
long gyro_x_cal, gyro_y_cal, gyro_z_cal;
boolean set_gyro_angles;

long acc_x, acc_y, acc_z, acc_total_vector;
float angle_roll_acc, angle_pitch_acc;

float angle_pitch, angle_roll;
int angle_pitch_buffer, angle_roll_buffer;
float angle_pitch_output, angle_roll_output;

// Setup timers and temp variables
long loop_timer;
int temp;

// Display counter
int displaycount = 0

void setup() \{

  //Start I2C
  Wire.begin();


  //Setup the registers of the MPU-6050
  setup_mpu_6050_registers();

  //Read the raw acc and gyro data from the MPU-6050 1000 times
  for (int cal_int = 0; cal_int < 1000 ; cal_int ++)
  {
    read_mpu_6050_data();
    //Add the gyro x offset to the gyro_x_cal variable
    gyro_x_cal += gyro_x;
```

```
    //Add the gyro y offset to the gyro_y_cal variable
    gyro_y_cal += gyro_y;
    //Add the gyro z offset to the gyro_z_cal variable
    gyro_z_cal += gyro_z;
    //Delay 3us to have 250Hz for-loop
    delay(3);
  }

  // Divide all results by 1000 to get average offset\\
  gyro_x_cal /= 1000;
  gyro_y_cal /= 1000;
  gyro_z_cal /= 1000;

  // Start Serial Monitor
  Serial.begin(115200);

  // Init Timer
  loop_timer = micros();
}

void loop()\{

  // Get data from MPU-6050
  read_mpu_6050_data();

  //Subtract the offset values from the raw gyro values
  gyro_x -= gyro_x_cal;
  gyro_y -= gyro_y_cal;
  gyro_z -= gyro_z_cal;

  //Gyro angle calculations . Note 0.0000611 = 1 / (250Hz x 65.5)

  //Calculate the traveled pitch angle and add this to the angle_pitch
    variable

  angle_pitch += gyro_x * 0.0000611;
  //Calculate the traveled roll angle and add this to the angle_roll
    variable\
  //0.000001066 = 0.0000611 * (3.142(PI) / 180degr) The Arduino sin
    function is in radians
  angle_roll += gyro_y * 0.0000611;

  //If the IMU has yawed transfer the roll angle to the pitch angle
  angle_pitch += angle_roll * sin(gyro_z * 0.000001066);
  //If the IMU has yawed transfer the pitch angle to the roll angle
  angle_roll -= angle_pitch * sin(gyro_z * 0.000001066);

  //Accelerometer angle calculations

  //Calculate the total accelerometer vector\\
  acc_total_vector = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));
```

```
  //57.296 = 1 / (3.142 / 180) The Arduino asin function is in radians
  //Calculate the pitch angle
  angle_pitch_acc = asin((float)acc_y/acc_total_vector)* 57.296;
  //Calculate the roll angle     \\
  angle_roll_acc = asin((float)acc_x/acc_total_vector)* -57.296;

  //Accelerometer calibration value for pitch
  angle_pitch_acc -= 0.0;\\
  //Accelerometer calibration value for roll
  angle_roll_acc -= 0.0;

  if(set_gyro_angles)
  {

  //If the IMU has been running
  //Correct the drift of the gyro pitch angle with the accelerometer pitch angle
    angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;
    //Correct the drift of the gyro roll angle with the accelerometer roll angle
    angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;
  \}
  else\{
    //IMU has just started
    //Set the gyro pitch angle equal to the accelerometer pitch angle
    angle_pitch = angle_pitch_acc;
    //Set the gyro roll angle equal to the accelerometer roll angle
    angle_roll = angle_roll_acc;
    //Set the IMU started flag
    set_gyro_angles = true;
  \}

  //To dampen the pitch and roll angles a complementary filter is used\\
  //Take 90% of the output pitch value and add 10% of the raw pitch value \\
  angle_pitch_output = angle_pitch_output * 0.9 + angle_pitch * 0.1;\\
  //Take 90% of the output roll value and add 10% of the raw roll value \\
  angle_roll_output = angle_roll_output * 0.9 + angle_roll * 0.1;\\
  //Wait until the loop_timer reaches 4000us (250Hz) before starting the next loop

  // Print to Serial Monitor
  //Serial.print(" | Angle  = ");
  Serial.println(angle_pitch_output);


  // Increment the display counter
  displaycount = displaycount +1;


Serial.println(angle_pitch_output );
 while(micros() - loop_timer < 4000);
```

```
 //Reset the loop timer
 loop_timer = micros();

}

void setup_mpu_6050_registers()\{

  //Activate the MPU-6050

  //Start communicating with the MPU-6050
  Wire.beginTransmission(0x68);
  //Send the requested starting register
  Wire.write(0x6B);  \
  //Set the requested starting register
  Wire.write(0x00);
  //End the transmission
  Wire.endTransmission();

  //Configure the accelerometer (+/-8g)

  //Start communicating with the MPU-6050
  Wire.beginTransmission(0x68);
  //Send the requested starting register
  Wire.write(0x1C);
  //Set the requested starting register
  Wire.write(0x10);
  //End the transmission
  Wire.endTransmission();

  //Configure the gyro (500dps full scale)

  //Start communicating with the MPU-6050
  Wire.beginTransmission(0x68);
  //Send the requested starting register
  Wire.write(0x1B);
  //Set the requested starting register
  Wire.write(0x08);
  //End the transmission
  Wire.endTransmission();

}


void read_mpu_6050_data(){

  //Read the raw gyro and accelerometer data

  //Start communicating with the MPU-6050
  Wire.beginTransmission(0x68);
  //Send the requested starting register
  Wire.write(0x3B);
  //End the transmission
```

```
  Wire.endTransmission();
  //Request 14 bytes from the MPU-6050
  Wire.requestFrom(0x68,14);
  //Wait until all the bytes are received
  while(Wire.available() < 14);

  //Following statements left shift 8 bits, then bitwise OR.
  //Turns two 8-bit values into one 16-bit value
  acc_x = Wire.read()<<8|Wire.read();
  acc_y = Wire.read()<<8|Wire.read();
  acc_z = Wire.read()<<8|Wire.read();
  temp = Wire.read()<<8|Wire.read();
  gyro_x = Wire.read()<<8|Wire.read();
  gyro_y = Wire.read()<<8|Wire.read();
  gyro_z = Wire.read()<<8|Wire.read();
}
```

1. .4 Servo

Unlike IMU and BMP servo motors are very easy to use. The connection is as shown in figure 9.

In this project we will use the servo motor as a trigger for parachute deployment system.This part will be explained in detail in the rest part of the project.Hence be sure that there is no error in the servo.

Code of servo for testing is given below:

```


#include<Servo.h>
int ServoPin=8;
void setup()
\{Serial.begin(9600);
Servo servo;
servo.attach(8);}
void main()
{ for( int i=0;i<=180;i++)
{servo.write(i);
delay(100);
}
```

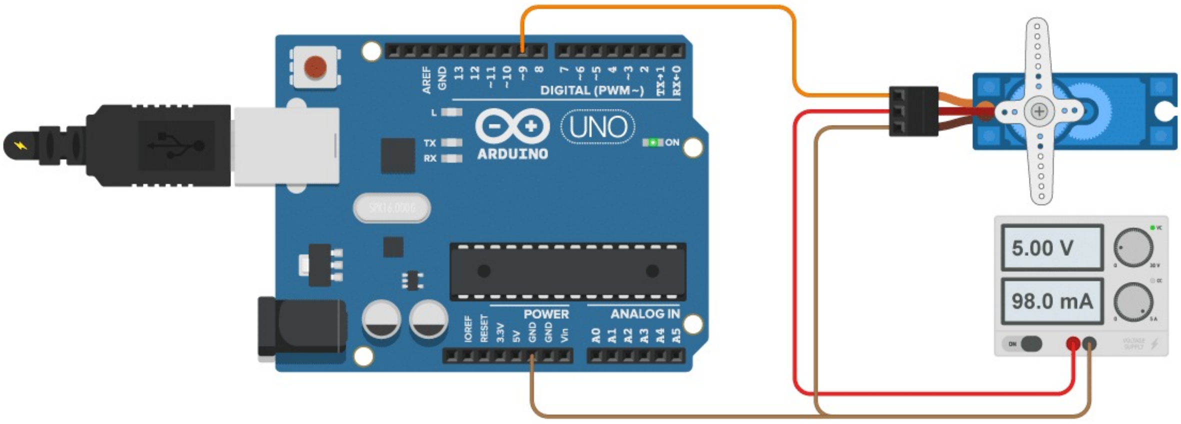


Figure 9: Caption

---

1. .5 Testing the barometer and servo system

After complete testing of each electronics now it's to check the BMP180 and servo all together are woking properly or not. system is arranged as shown in fig 10.

The results of the testing is shown in figure 11,here if the height measured by BMP180 is more than 1 meter then servo motor will rotate by an angle 180.
code of the following test:

```


#include <SFE_BMP180.h>
#include <Wire.h>
#include<Servo.h>
SFE_BMP180 pressure;
double baseline; // baseline pressure
Servo servo;
void setup()
{
  Serial.begin(115200);
  Serial.println("REBOOT");
```

```
 servo.attach(8);

  if (pressure.begin())
    Serial.println("BMP180 init success");
  else
  {
    Serial.println("BMP180 init fail (disconnected?)\n\n");
    while(1); // Pause forever.
  }


  baseline = getPressure();



}
void loop()
{
  double a,P;


  P = getPressure();
  a = pressure.altitude(P,baseline);

  Serial.print("relative altitude: ");
  if (a >= 0.0) Serial.print(" ");
  Serial.print(a,1);
  Serial.println(" meters");

  delay(1500);
 if(a>=1.0)
 {
  servo.write(180);
  Serial.println("servo angle is 180");
  delay(1000);
 }
 else\\
 {  Serial.println("servo angle is 0");
 delay(1500);
 }
 }




double getPressure()
{
  char status;
  double T,P,p0,a;


  status = pressure.startTemperature();
  if (status != 0)
```

```
  {

    delay(status);


    status = pressure.getTemperature(T);
    if (status != 0)
    {

      status = pressure.startPressure(3);
      if (status != 0)
      {

        delay(status);


        status = pressure.getPressure(P,T);
        if (status != 0)
        {
          return(P);
        }

        else Serial.println("error retrieving pressure measurement\n");
      }
      else Serial.println("error starting pressure measurement\n");
    }
    else Serial.println("error retrieving temperature measurement\n");

  }
  else Serial.println("error starting temperature measurement\n");
}
```

Once it is done control system is almost ready.

1. **circuit diagram for the final system:**

All the circuit diagram and connections of the sensors are shown in figures 12-13:

-

5 MECHANICAL DESIGN OF DEPLOYMENT SYSTEM

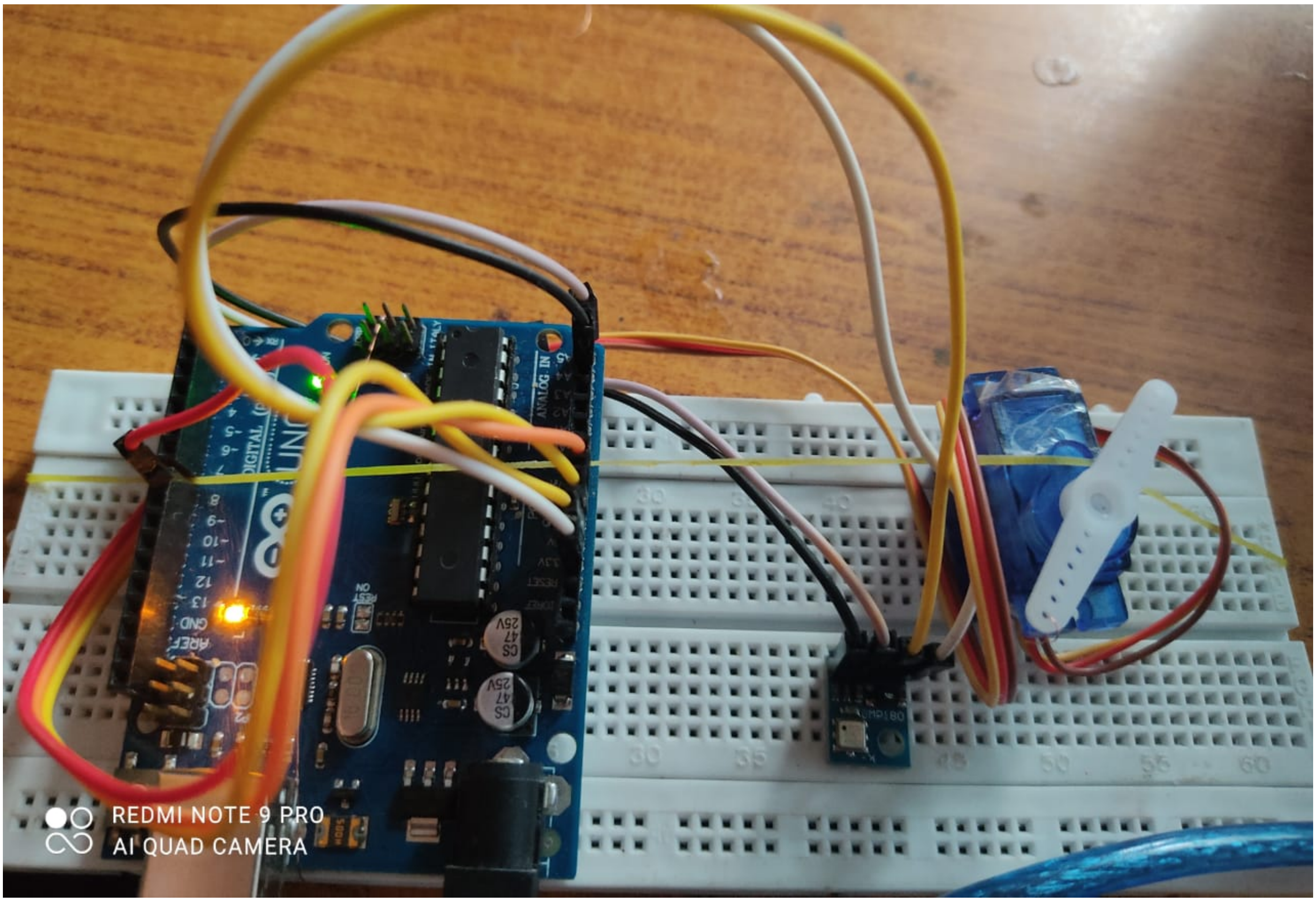


Figure 10: servo_using_BMP180

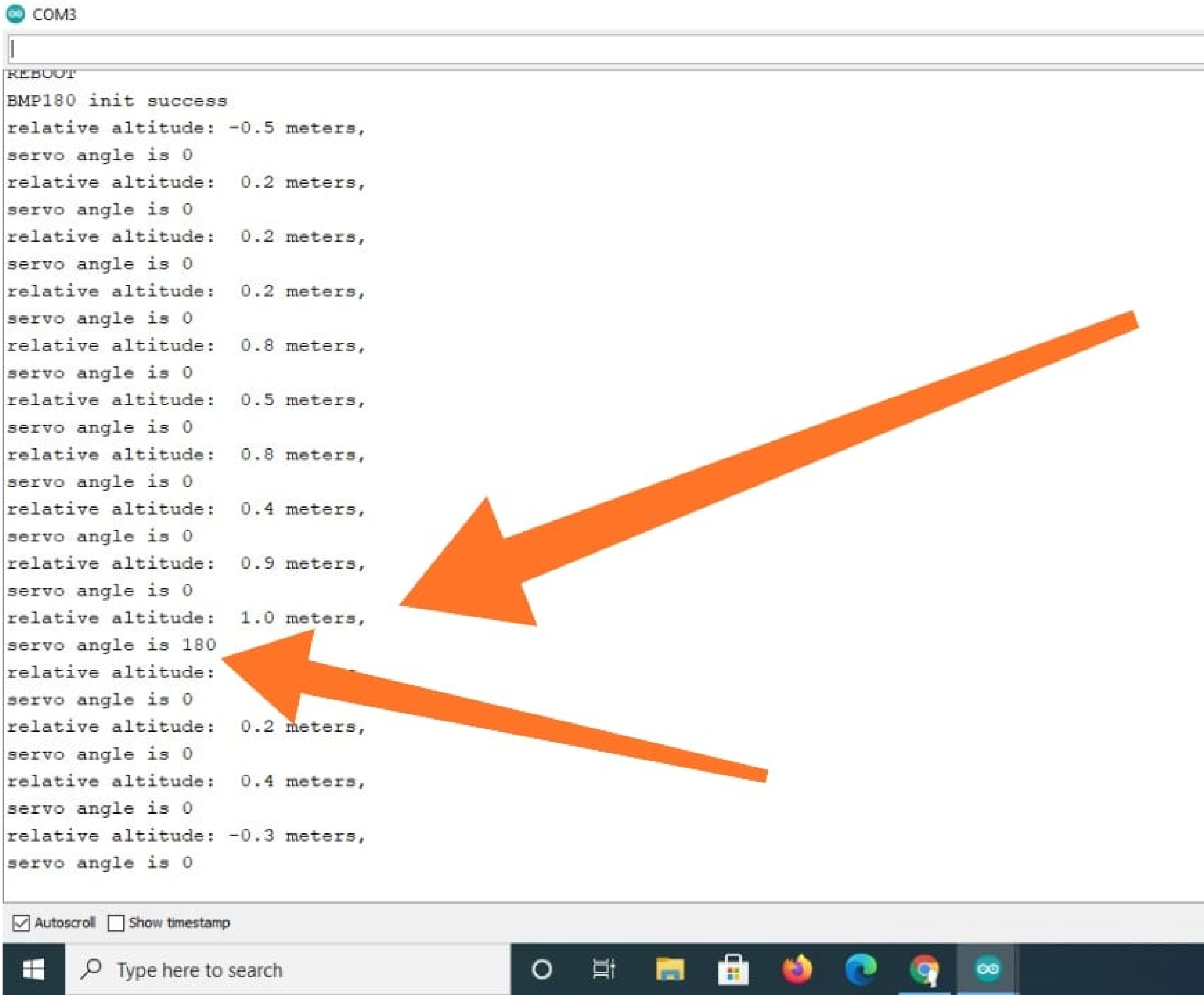


Figure 11: REsults

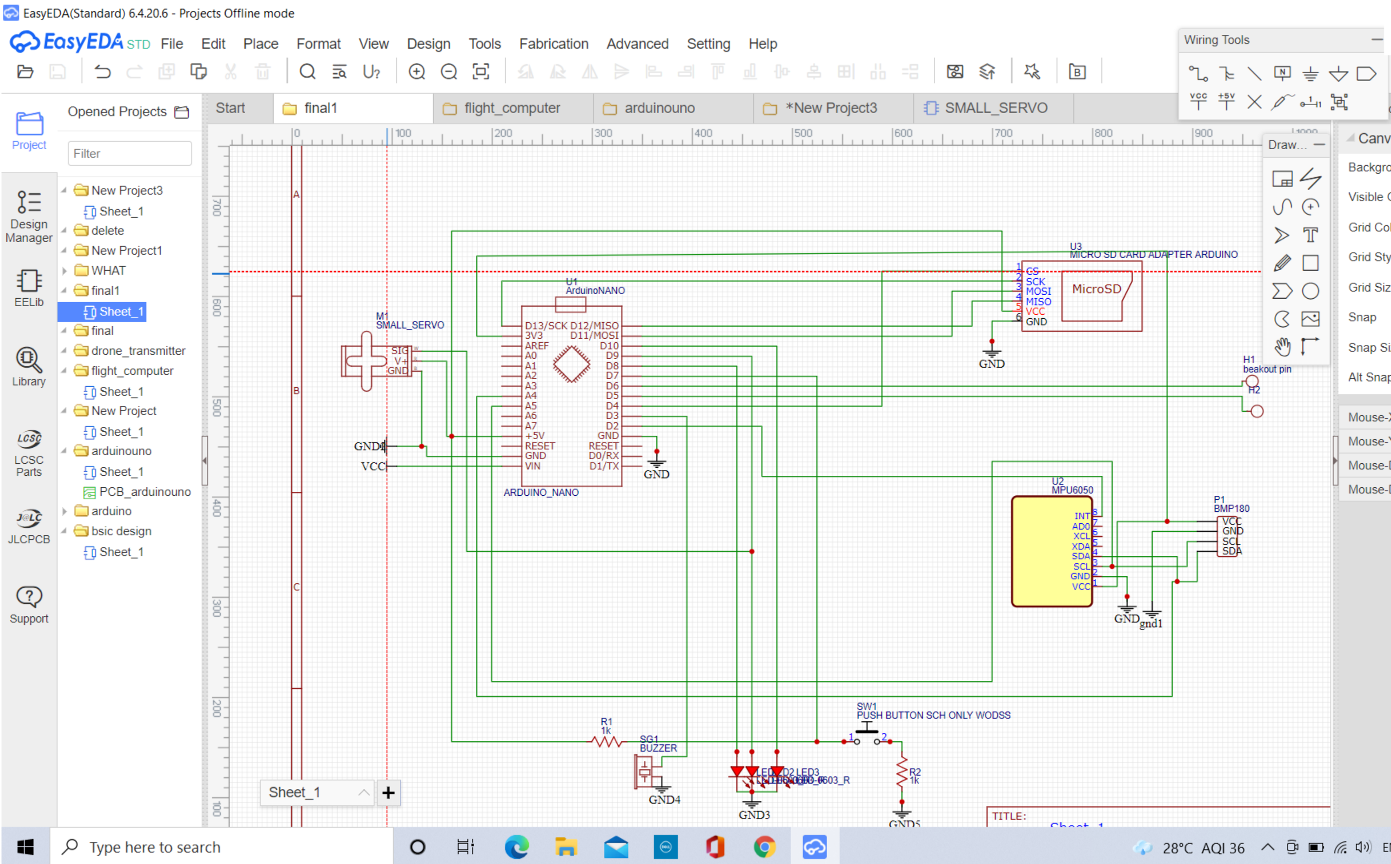
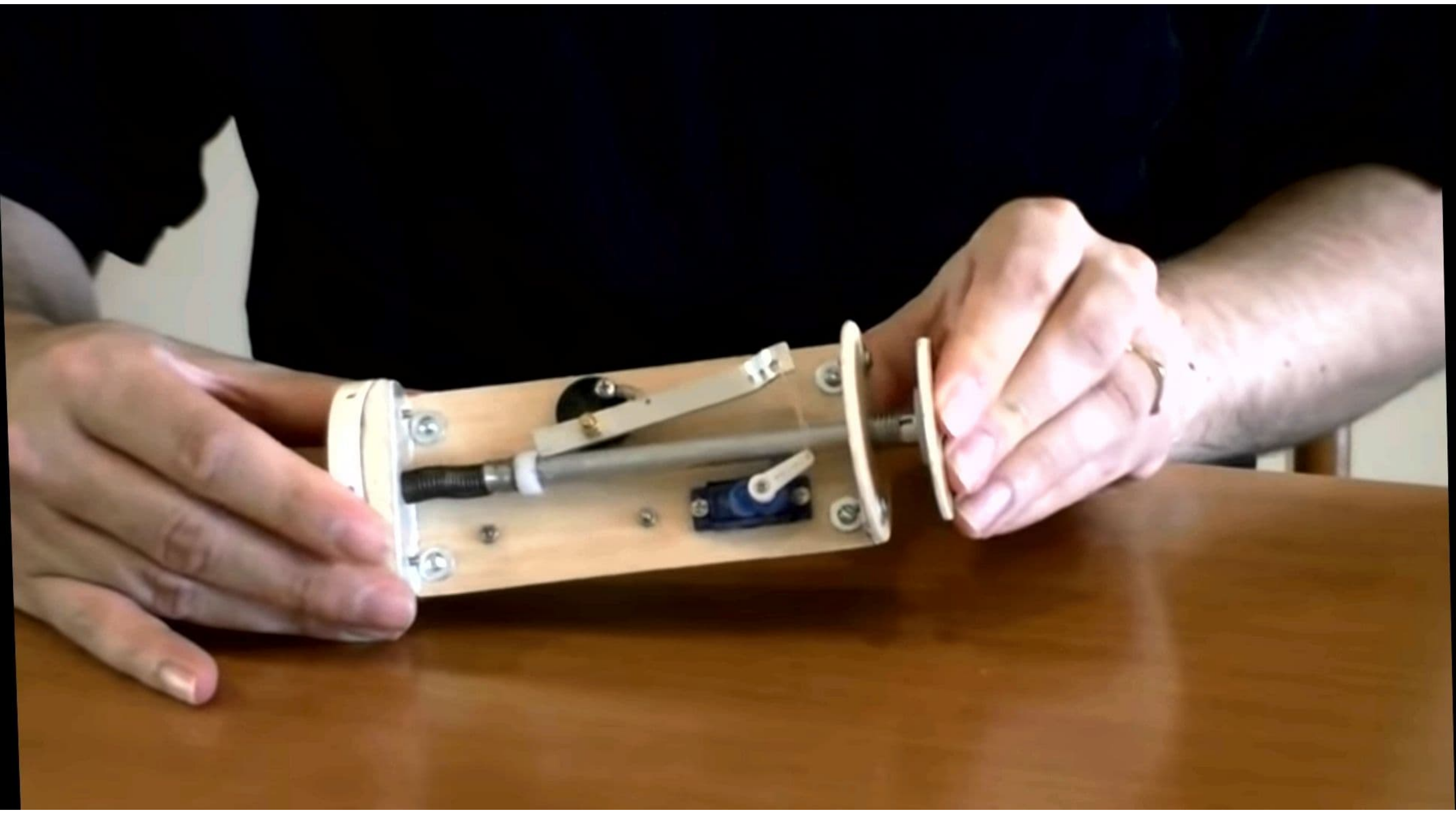
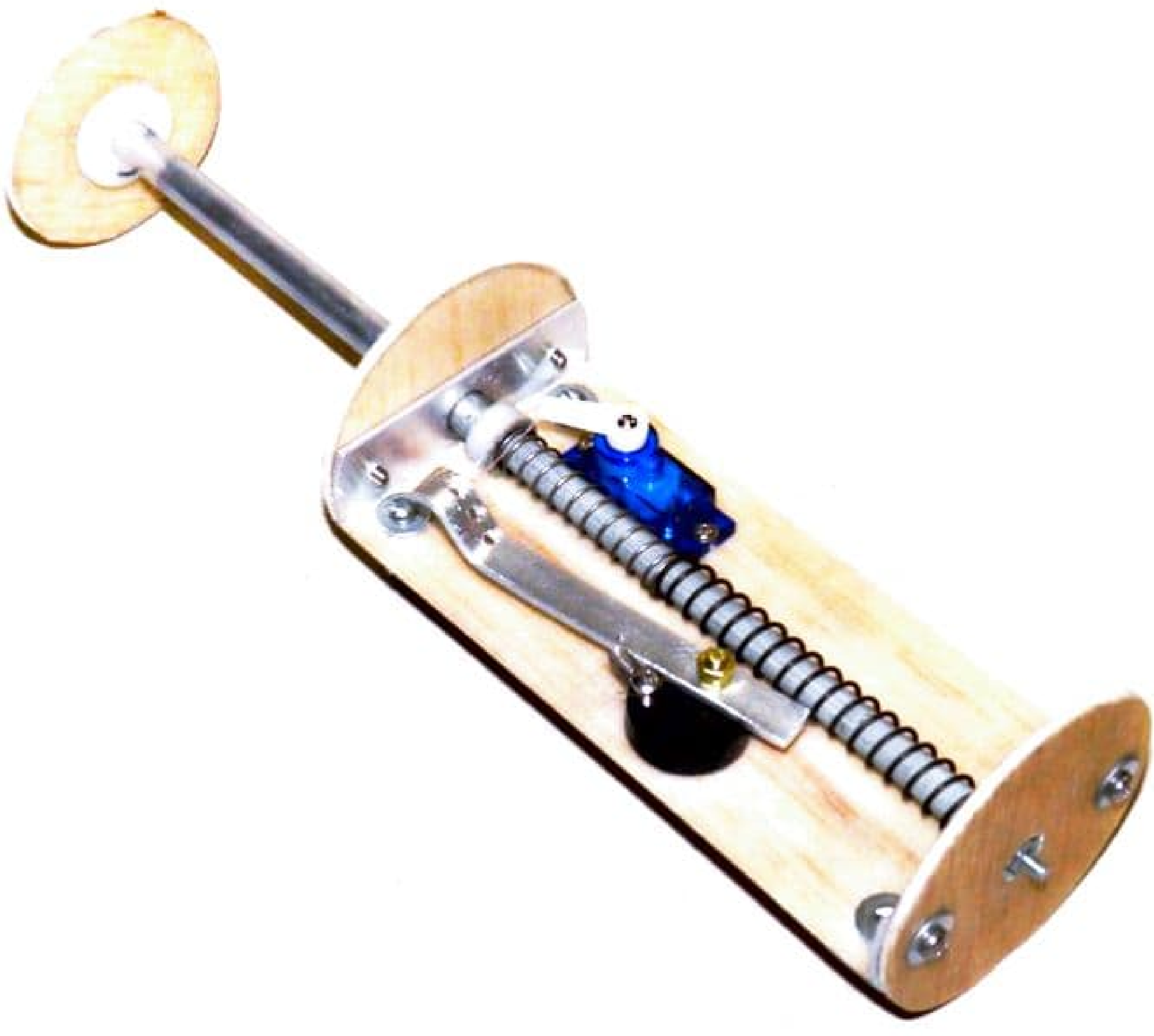


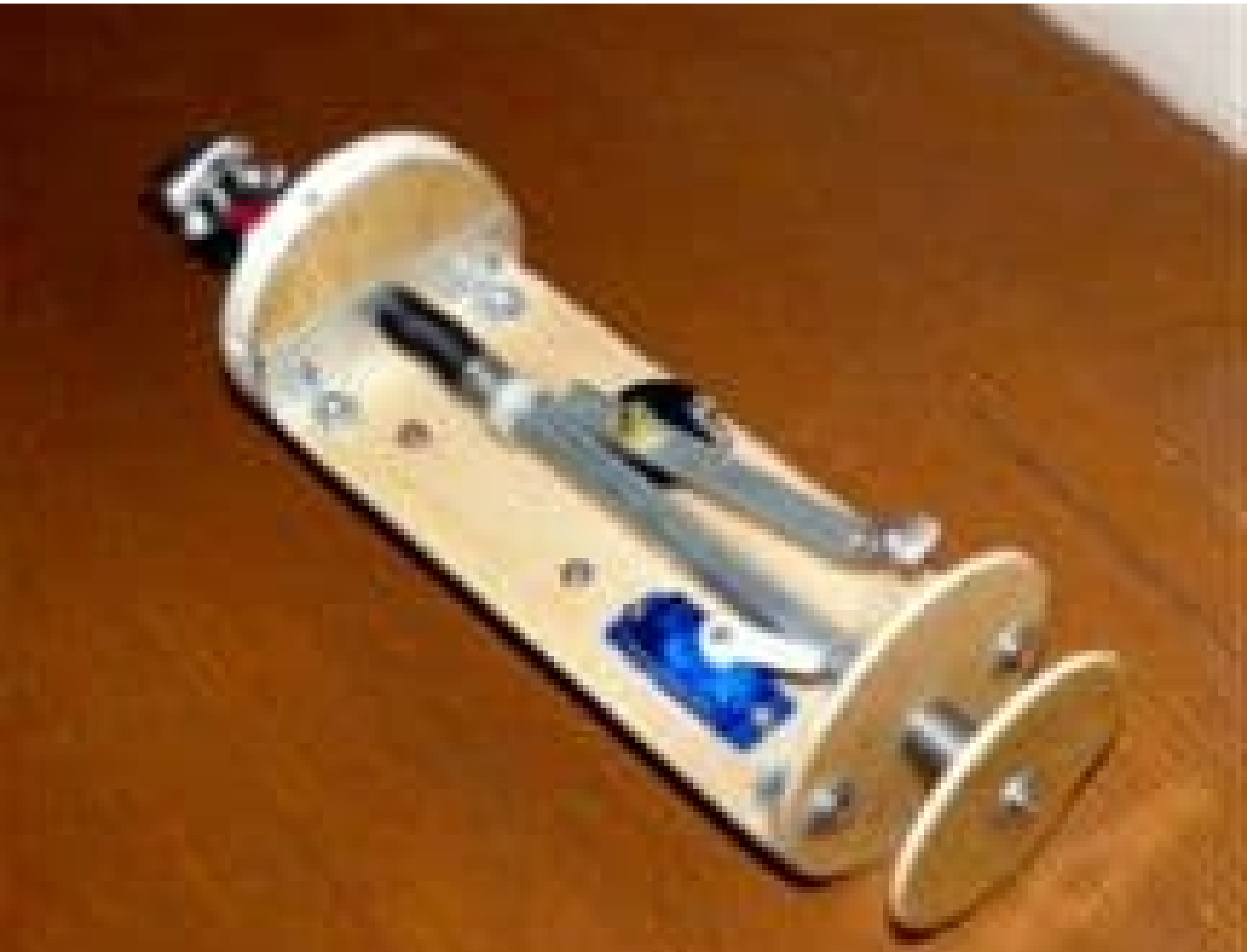
Figure 12: all the connections of sensors

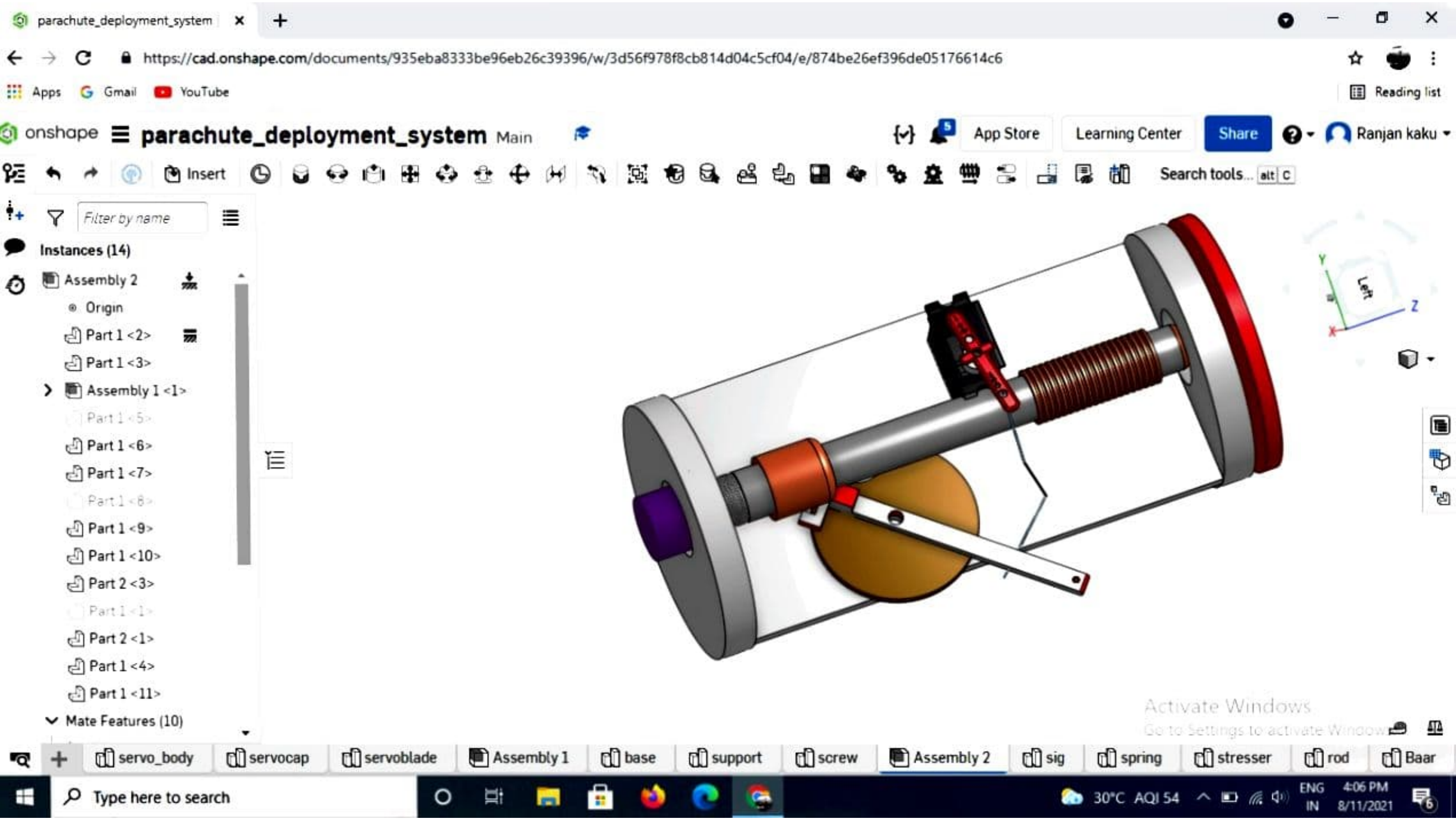
After a brief testing of our electronics ,Now t is time to understand how and where we can use this.

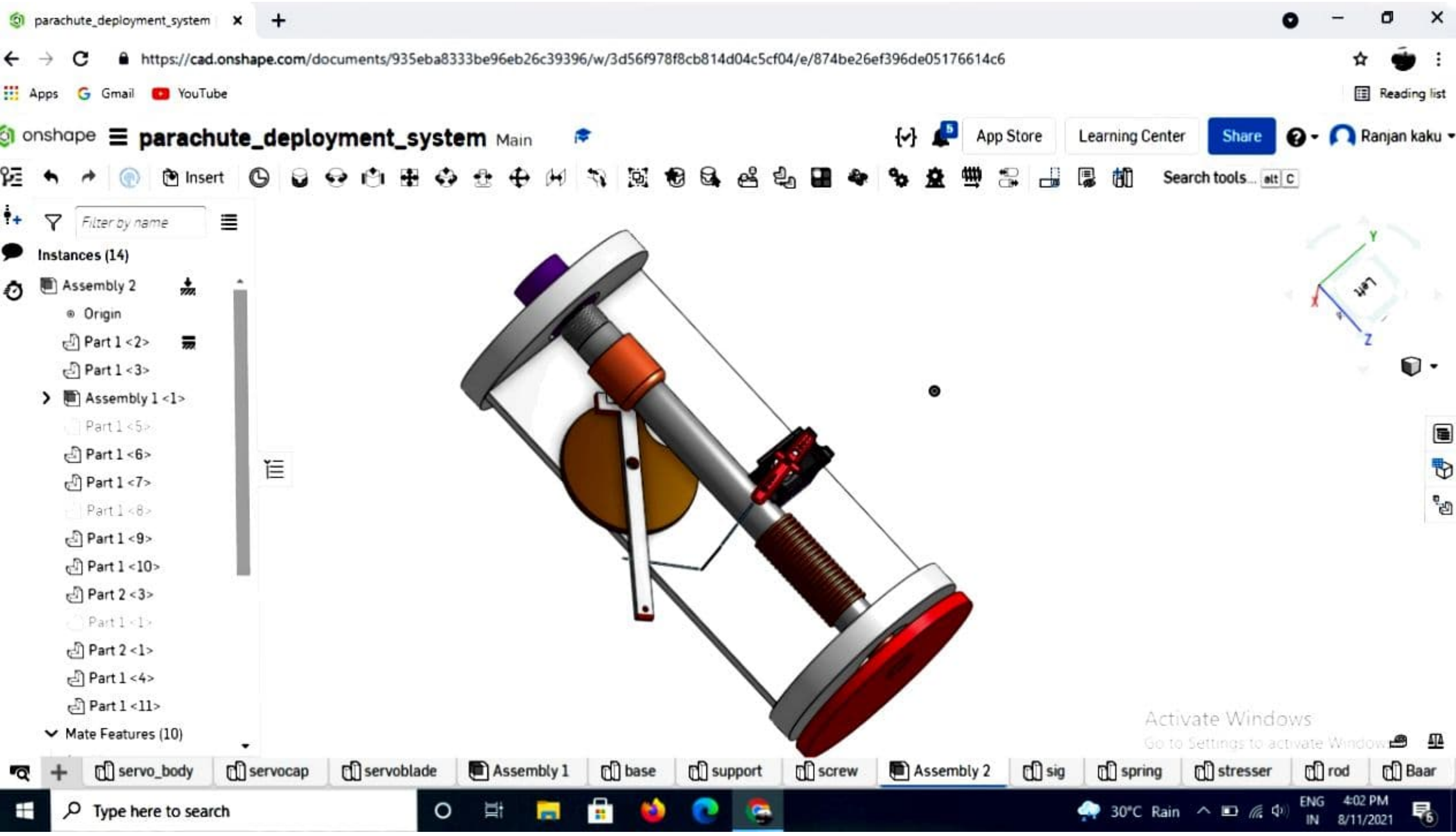
In the deployment of the parachute it is very very important that it never fails.There are lots of method for the deployment of parachute,but the method I an going to use I think is the most secure ,belivable as well as easy method...... You can go details to the mechanical system of the deployment system that we are going to use in the pdf given below.—-

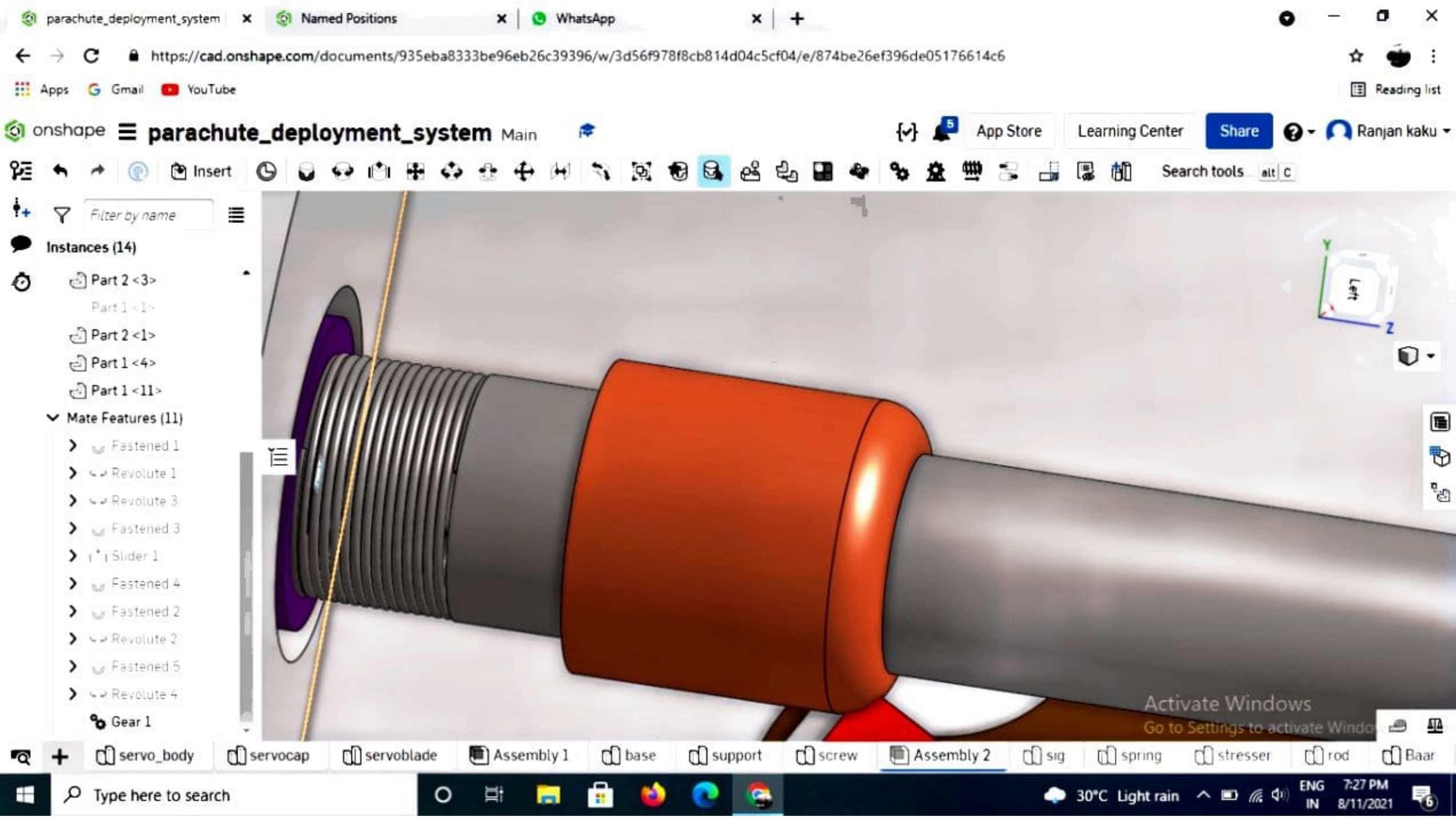


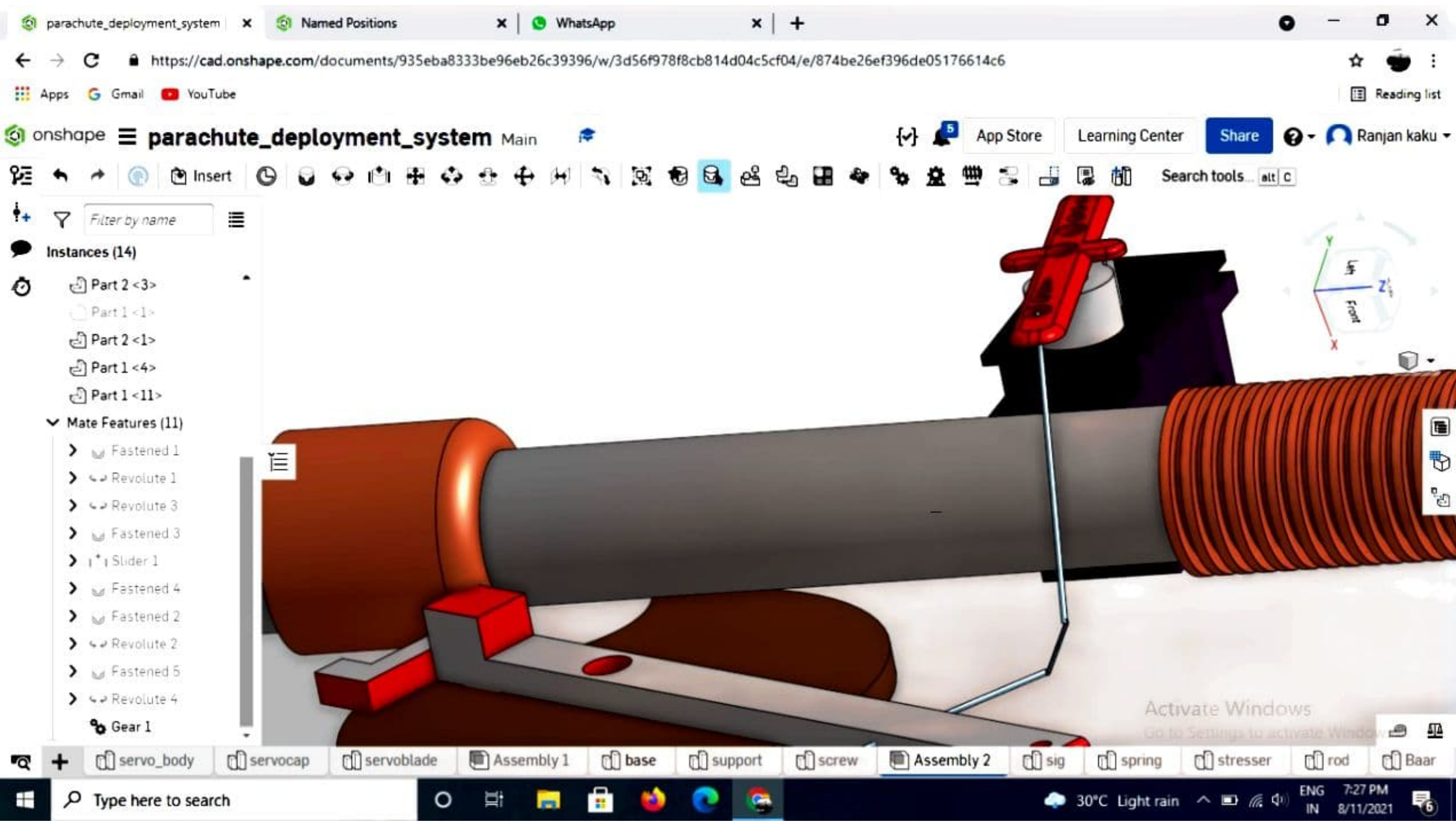





parachute_deployment_system
https://cad.onshape.com/documents/935eba8333be96eb26c39396/w/3d56f978f8cb814d04c5cf04/e/874be26ef396de05176614c6
Apps
Gmail
YouTube
Reading list
onshape
parachute_deployment_system
Main
App Store
Learning Center
Share
Ranjan kaku
Insert
Search tools...
Filter by name
Instances (14)
Assembly 2
Origin
Part 1 <2>
Part 1 <3>
Assembly 1 <1>
Part 1 <5>
Part 1 <6>
Part 1 <7>
Part 1 <8>
Part 1 <9>
Part 1 <10>
Part 2 <3>
Part 1 <1>
Part 2 <1>
Part 1 <4>
Part 1 <11>
Mate Features (10)
Left
Activate Windows
Go to Settings to activate Windows.
servo_body
servocap
servoblade
Assembly 1
base
support
screw
Assembly 2
sig
spring
stresser
rod
Baar
Type here to search
30°C AQI 54
ENG
IN
4:06 PM
8/11/2021


parachute_deployment_system
https://cad.onshape.com/documents/935eba8333be96eb26c39396/w/3d56f978f8cb814d04c5cf04/e/874be26ef396de05176614c6
Apps
Gmail
YouTube
Reading list
onshape
parachute_deployment_system Main
App Store
Learning Center
Share
Ranjan kaku
Insert
Search tools...
Filter by name
Instances (14)
Assembly 2
Origin
Part 1 <2>
Part 1 <3>
Assembly 1 <1>
Part 1 <5>
Part 1 <6>
Part 1 <7>
Part 1 <8>
Part 1 <9>
Part 1 <10>
Part 2 <3>
Part 1 <1>
Part 2 <1>
Part 1 <4>
Part 1 <11>
Mate Features (10)
Activate Windows
servo_body
servocap
servoblade
Assembly 1
base
support
screw
Assembly 2
sig
spring
stresser
rod
Baar
Type here to search
30°C Rain
ENG
IN
4:02 PM
8/11/2021


parachute_deployment_system
Named Positions
WhatsApp
https://cad.onshape.com/documents/935eba8333be96eb26c39396/w/3d56f978f8cb814d04c5cf04/e/874be26ef396de05176614c6
Apps
Gmail
YouTube
Reading list
onshape
parachute_deployment_system
Main
App Store
Learning Center
Share
Ranjan kaku
Insert
Search tools
Filter by name
Instances (14)
Part 2 <3>
Part 1 <1>
Part 2 <1>
Part 1 <4>
Part 1 <11>
Mate Features (11)
Fastened 1
Revolute 1
Revolute 3
Fastened 3
Slider 1
Fastened 4
Fastened 2
Revolute 2
Fastened 5
Revolute 4
Gear 1
Activate Windows
Go to Settings to activate Windows
servo_body
servocap
servoblade
Assembly 1
base
support
screw
Assembly 2
sig
spring
stresser
rod
Baar
Type here to search
30°C Light rain
ENG
IN
7:27 PM
8/11/2021



parachute_deployment_system
Named Positions
WhatsApp
https://cad.onshape.com/documents/935eba8333be96eb26c39396/w/3d56f978f8cb814d04c5cf04/e/874be26ef396de05176614c6
Apps
Gmail
YouTube
Reading list
onshape
parachute_deployment_system
Main
App Store
Learning Center
Share
Ranjan kaku
Insert
Search tools...
Filter by name
Instances (14)
Part 2 <3>
Part 1 <1>
Part 2 <1>
Part 1 <4>
Part 1 <11>
Mate Features (11)
Fastened 1
Revolute 1
Revolute 3
Fastened 3
Slider 1
Fastened 4
Fastened 2
Revolute 2
Fastened 5
Revolute 4
Gear 1
Activate Windows
servo_body
servocap
servoblade
Assembly 1
base
support
screw
Assembly 2
sig
spring
stresser
rod
Baar
Type here to search
30°C Light rain
ENG
IN
7:27 PM
8/11/2021

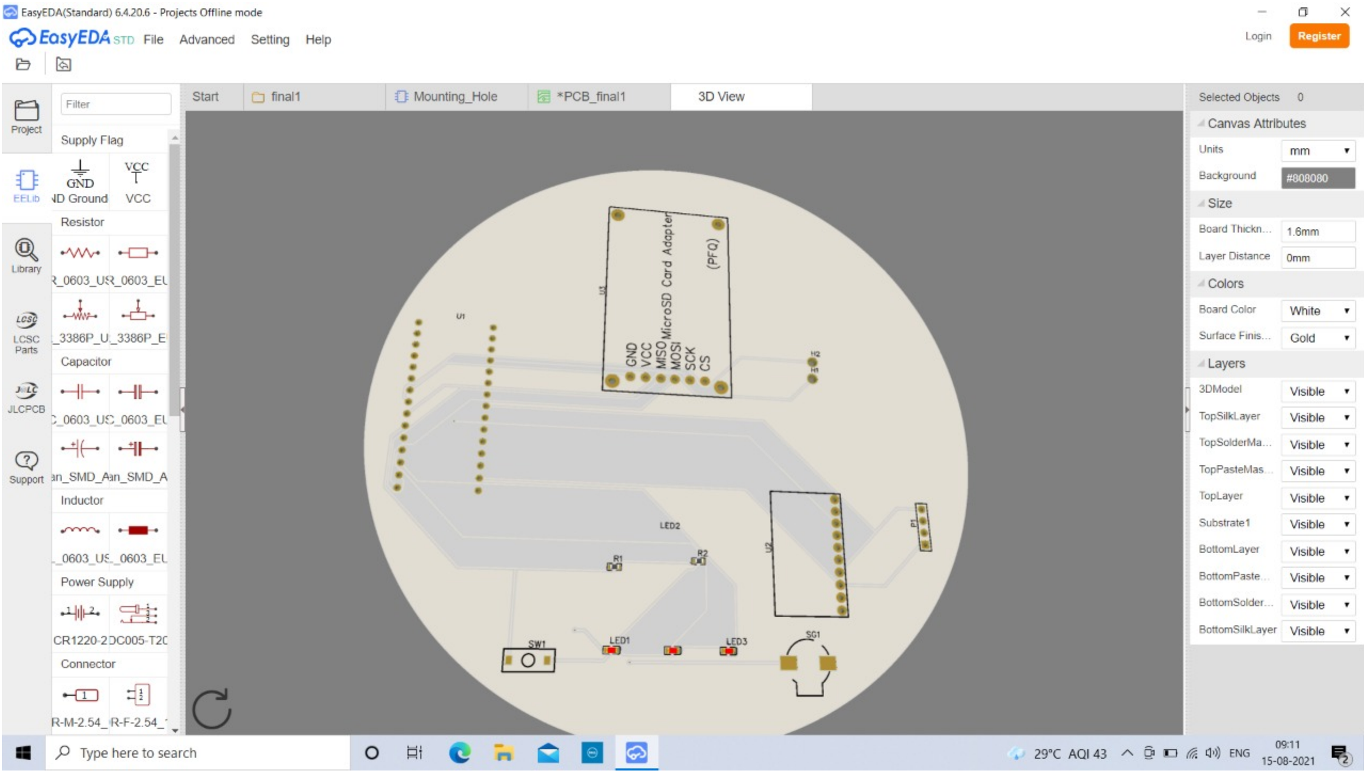


Figure 13: The circuit board for the flight computer

In this system we will compress the spring and the servo motor will act like a trigger.According to the command from BMP180 the servo motor will release the spring and parachute which is situated at the front ,will be shoot out.

After completing the system we are almost done with control system of the rocket.

6 Mechanical Structure Of the rocket

1. Stability

How fine is your rocket is,how sensitive is your control system, all are meaningless if your rocket isn't stable.Hence stability is the most important part for all kinds of rockets.
Simply said **'A rocket is stable if the centre of gravity is ahead of centre of pressure'**.

But what this term refers and how to calculate them?
Any body moving in a fluid experiences pressure forces over its surface. The concepts of center of pressure, aerodynamics center and neutral point are useful in understanding the effects of these forces. Let's take an airfoil moving in air with subsonic flow attached to the body.(figure 14)

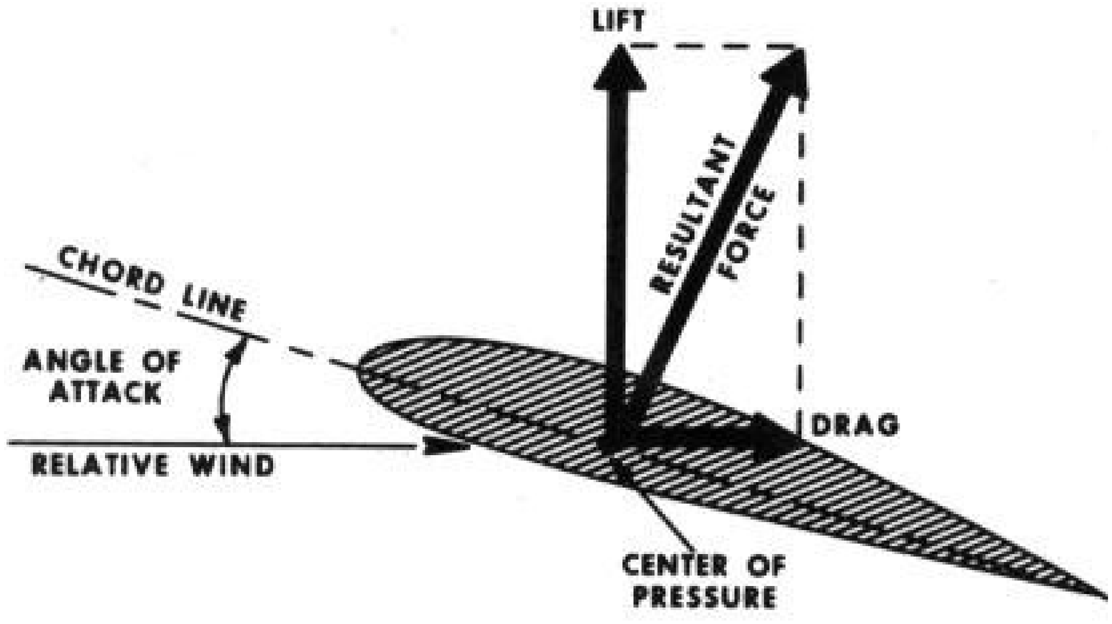


Figure 14: forces on a aeroplane wings

Center of Pressure The center of pressure is the point where the total sum of a pressure field acts on a body. In aerospace, this is the point on the airfoil (or wing) where the resultant vector (of lift and drag) acts.

How to calculate the center of mass and center of pressure

CENTER OF MASS:

Calculating center of mass is really easy.
For a general shaped object, there is a simple mechanical way to determine the center of gravity:

If we just balance the object using a string or an edge, the point at which the object is balanced is the center of gravity. (Just like balancing a pencil on your finger!)

CENTER OF PRESSURE Calculating center of pressure isn't as easy as Cg.Although there are various methods like wing tunnel to mathematics,but either they are too expansive,time consuming or really difficult to do. But the method I going yo explain is really easy although not perfectly accurate.This method is known as cardboard cutting method.

In this method you place your rocket on a flat cardboard and draw the boundary of 3D rocket on a 2D plane i.e projection . After this you cut out that part from the card board.
Figure 15 shows the method.

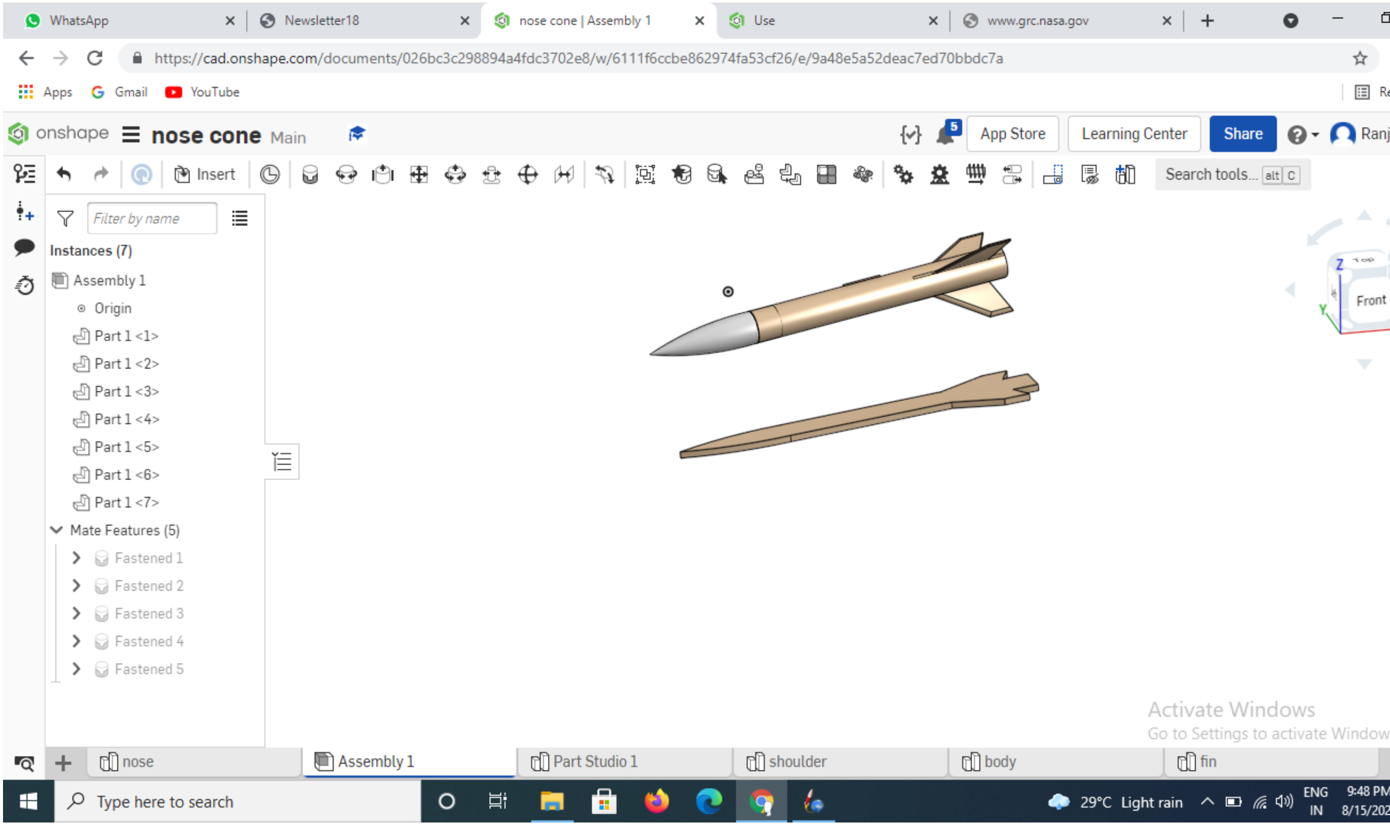


Figure 15: projection on a card board

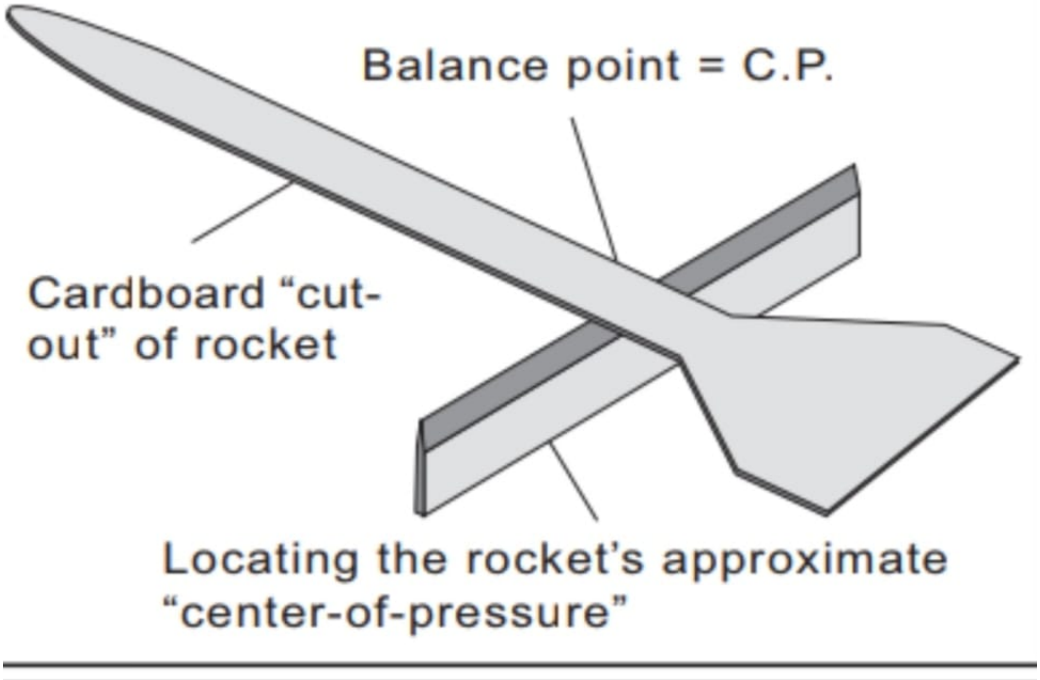


Figure 16: balancing point

At the final step,you need to balance the card board piece on your finger.The balancing point is the **Center of pressure**

If the center of pressure is ahead of center of gravity,it is unstable .Make sure that center of pressure is behind the center of mass.If not then try put some additional mass to the rocket such that center of gravity(C_g)*leadsCenterofpressure*(C_p)

•

7 Design and analysis of different parts of a rocket

I have already mentioned the different parts of rocket but in this section I will mention how to improve their performance.

1. **OpenRocket**

For designing my rocket and check its performance I used the software Open Rocket.

1. **Nose Cone**

For aerodynamics ***Nose cone*** **shape is very very important.In figure 17 the drag coefficients of different nose cone is shown.**

Dimensions

For this project I used a nose cone of length=8.48 cm,
base diameter=2.48 cm,
inner diameter=2.3,
shape parameter=1
material=plastic

It is better to choose parabolic,Ogive or long Eliptical as they have lower drag coefficients .Hence less energy loss .

1. **Body**

Body design of a rocket may be vary according to it's purpose.For a model rocket it is generally a hollow cylinder.

Dimensions

aft diameter=2 and wall thickness 2 mm.
material=Cardboard. ***I have used a 13 cm body tube and a transition section whose—shape=conical***
length=3 cm
fore diameter=2.48cm

aft diameter=2 and wall thickness 2 mm.
material=Cardboard.

1. Fins

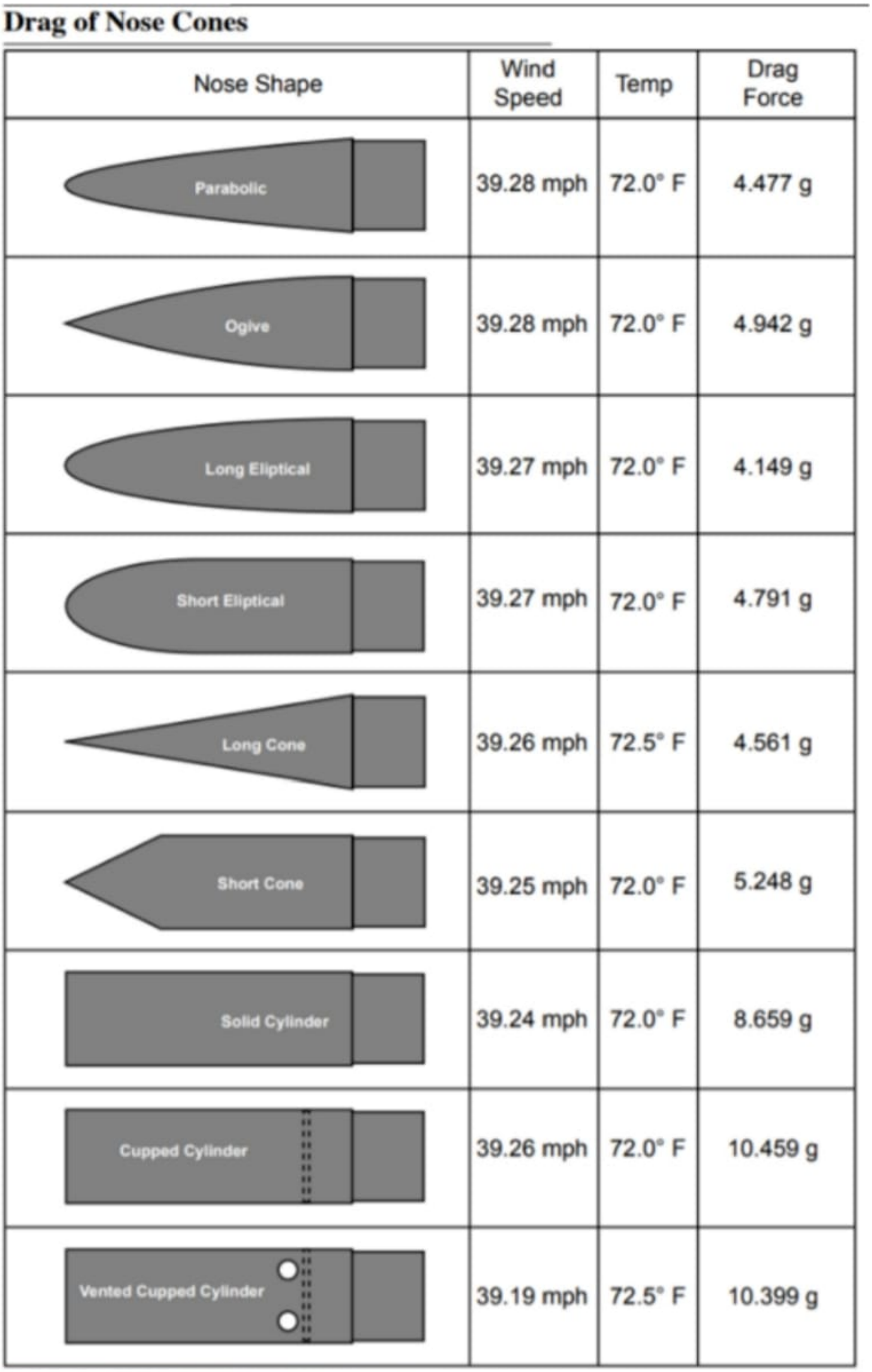
Fins is a very important part of a rocket.It is the reason why a rocket can fly vertically.Not only stability fins can also create a significant difference in the performance of a rocket.
There are different kinds of fins .FIG 18 .

Dimensions

IN this project I used elliptical shaped fins
number of fins=3
rotation =-38 deg;
Height=3 cm; root chord=6;
3.3 cm away from the bottom of the body tube. material=Balsa wood;

1. inner tube

Inner tube is used for placing rocket motor.It's size should be such that it is not too big to be heavy load for the rocket or not to small such that it can not fit a motor.



Drag of Nose Cones

Nose Shape	Wind Speed	Temp	Drag Force
Parabolic	39.28 mph	72.0° F	4.477 g
Ogive	39.28 mph	72.0° F	4.942 g
Long Eliptical	39.27 mph	72.0° F	4.149 g
Short Eliptical	39.27 mph	72.0° F	4.791 g
Long Cone	39.26 mph	72.5° F	4.561 g
Short Cone	39.25 mph	72.0° F	5.248 g
Solid Cylinder	39.24 mph	72.0° F	8.659 g
Cupped Cylinder	39.26 mph	72.0° F	10.459 g
Vented Cupped Cylinder	39.19 mph	72.5° F	10.399 g

Figure 17: Different nose cone and drag coefficient

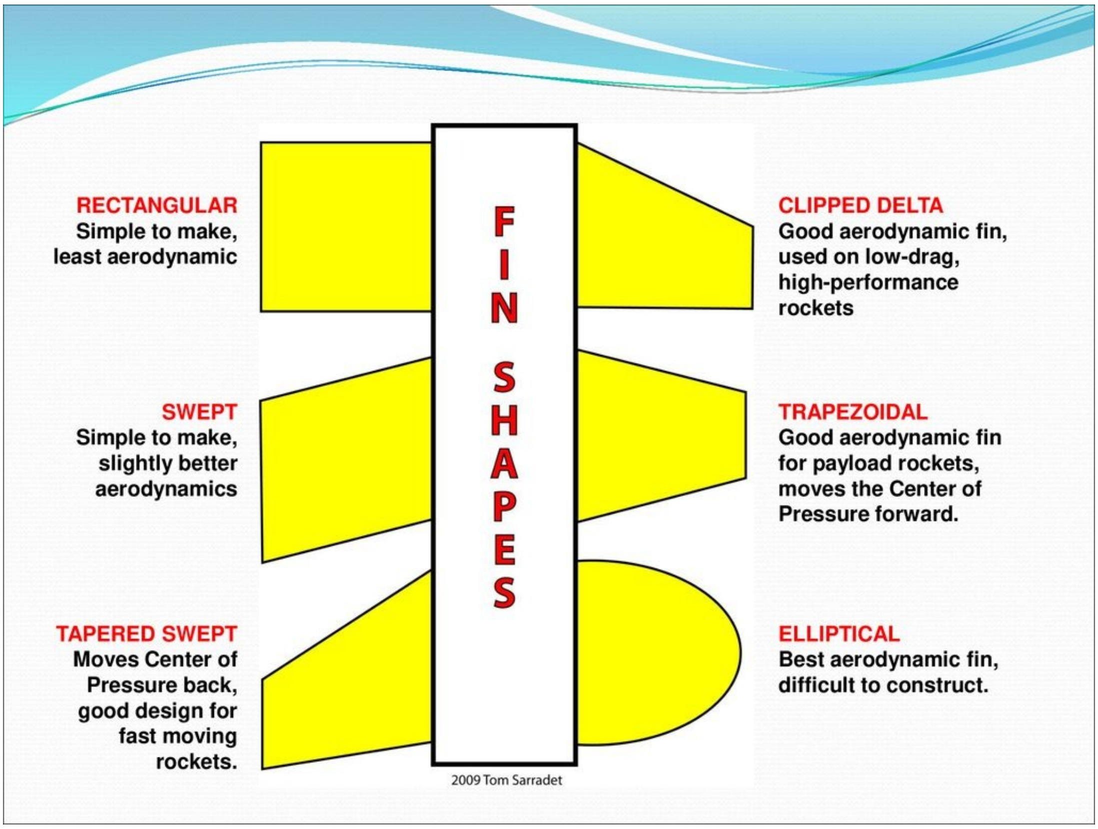


Figure 18: different fins

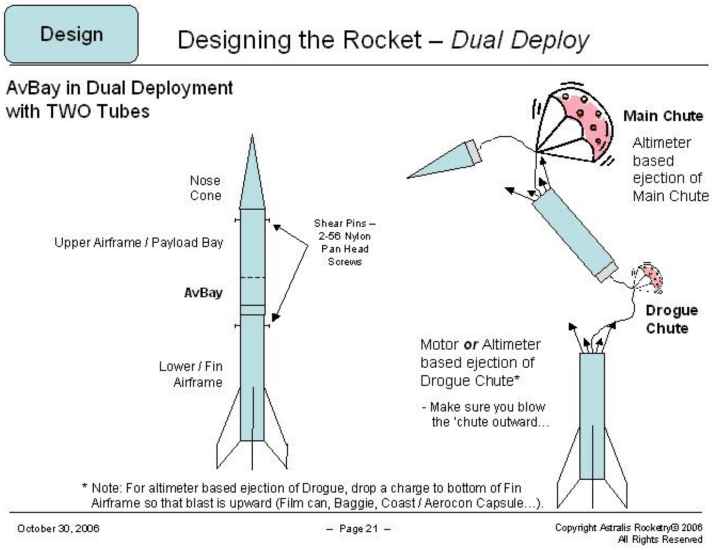


Figure 19: How a parachute opens

Dimensions
outer diameter =2 cm
inner diameter=1.9 cm
length=6 cm
material=PVC

1. Centering Ring

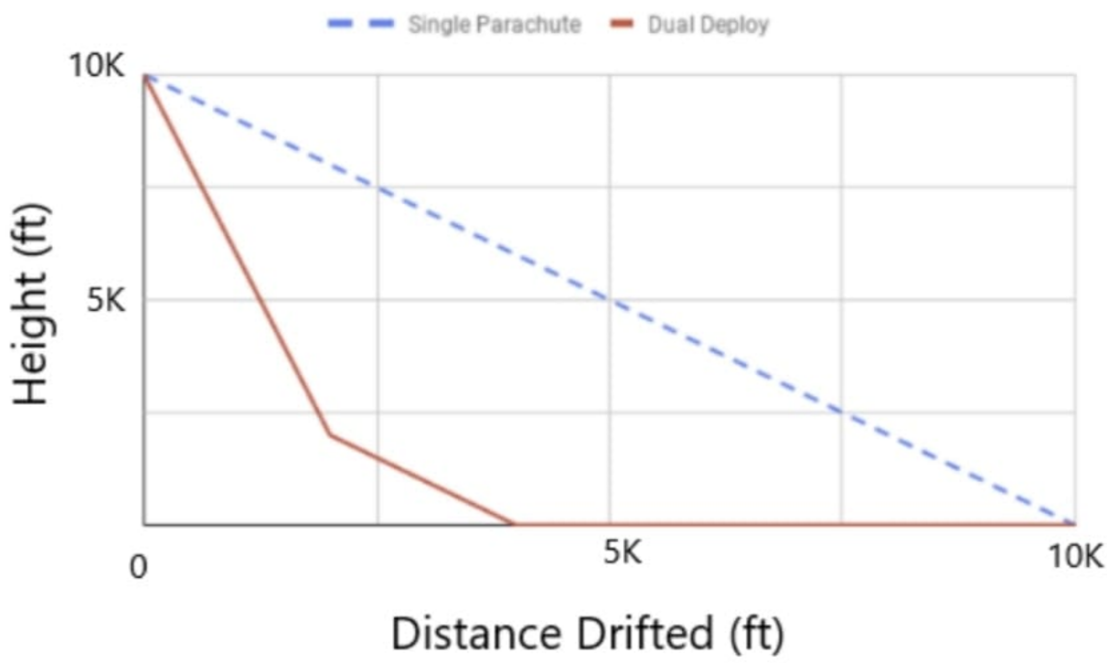
Centering ring is used for holding the *inner tube*.

Dimensions
Outer diameter=1.9
thickness=2.2 cm

1. *Parachute*

Parachute is also a very important part. It is the reason why a rocket lands safely. It reduces the velocity of the free falling rocket.
Depending on the number of parachute the rocket travels a distance along the horizon or ground level.
Generally a double parachute system is prefered over a single parachute since it reduce the rocket horizontal travel distance.However it increases the complexity of the deployment system and mass,hence higher risk and lower vertical travel distance of the rocket.A comparison in(fig 20)

Rocket Motor User should select a motor according his purpose.It is always recommended to use high quality factory made rocket motor.
for this project I used a D21 single use rocket engine.



Descent trajectories of rockets containing dual and single deployment systems.

Figure 20: graph of how effective is double parachute system

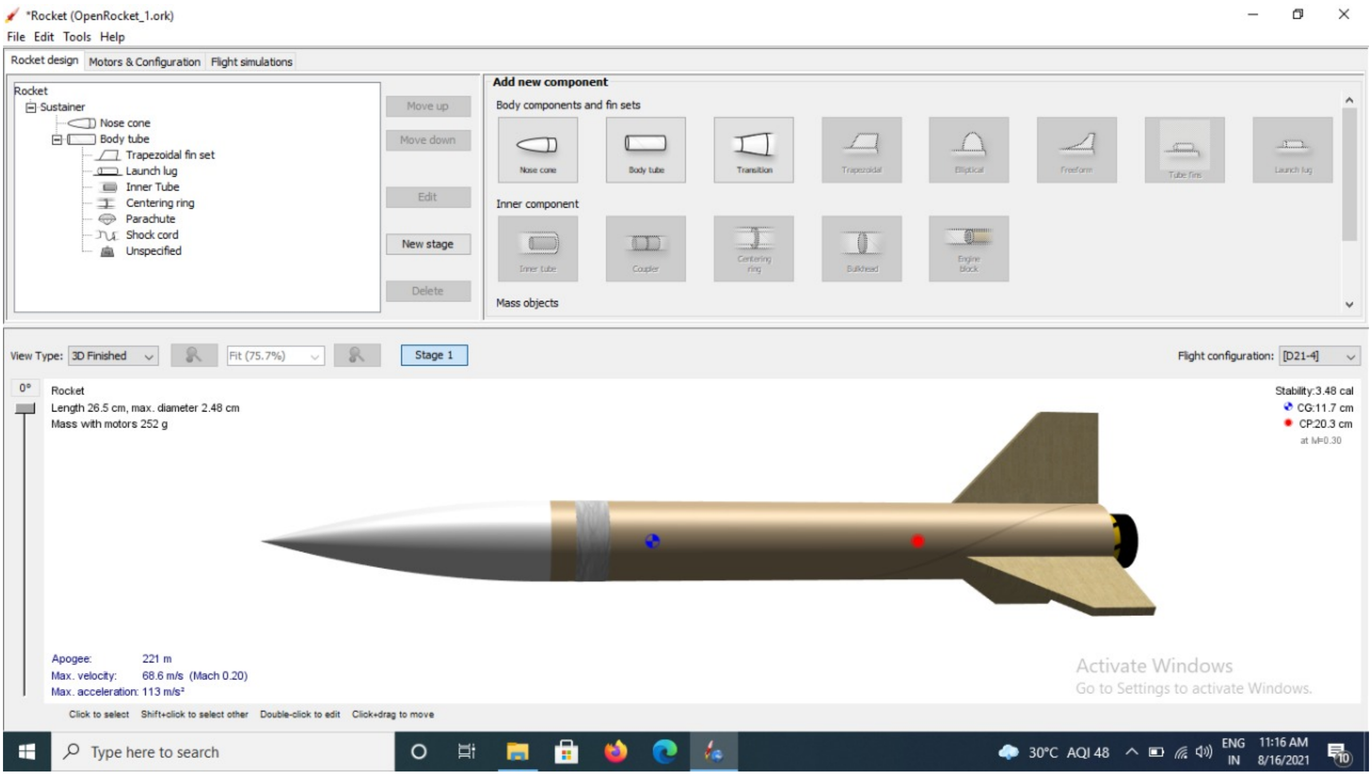


Figure 21: First design

1. **Analysis**

NOW its time to analysis the rocket and how can we improve them.
I have created two models of rockets having exactly same mass but some difference in structure.How ever I kept the engine same so that I can check which is more aerodynamics.
The design that I created earlier is in figure 21.

Clearly from fig 23 where I made a external aerodynamic simulation of the rocket I have seen the rocket structure can be improved specially by changing body and fins.

1. **improvement of The rocket design**

Fig 24-26 is the modified design of my rocket.As a lots of turbulance and pressure variation was found in the previous model ,hence I add a *Boat Tail.*Moreover I changed the fins shaped to elliptical.

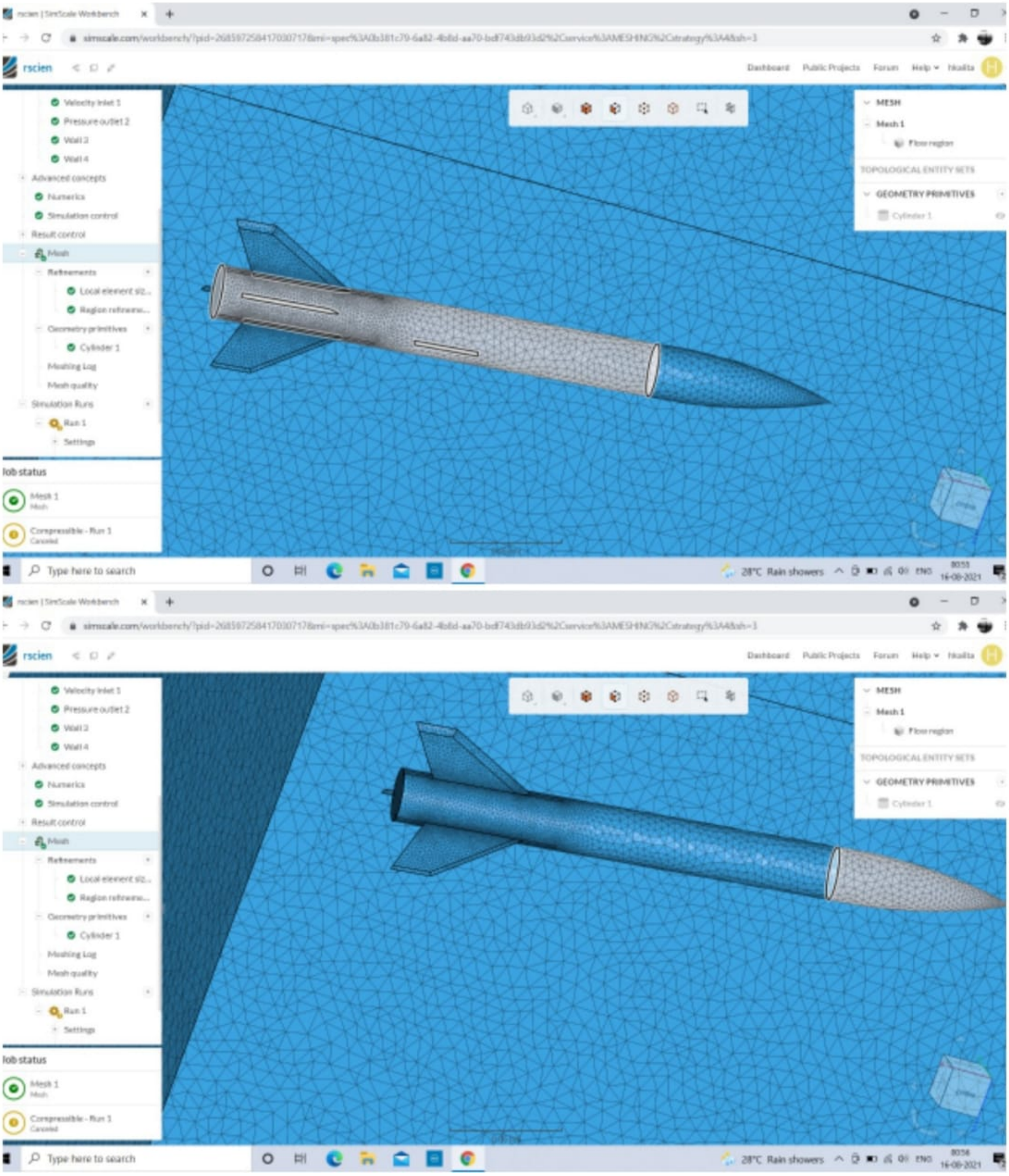


Figure 22: Mesh of the earlier rocket

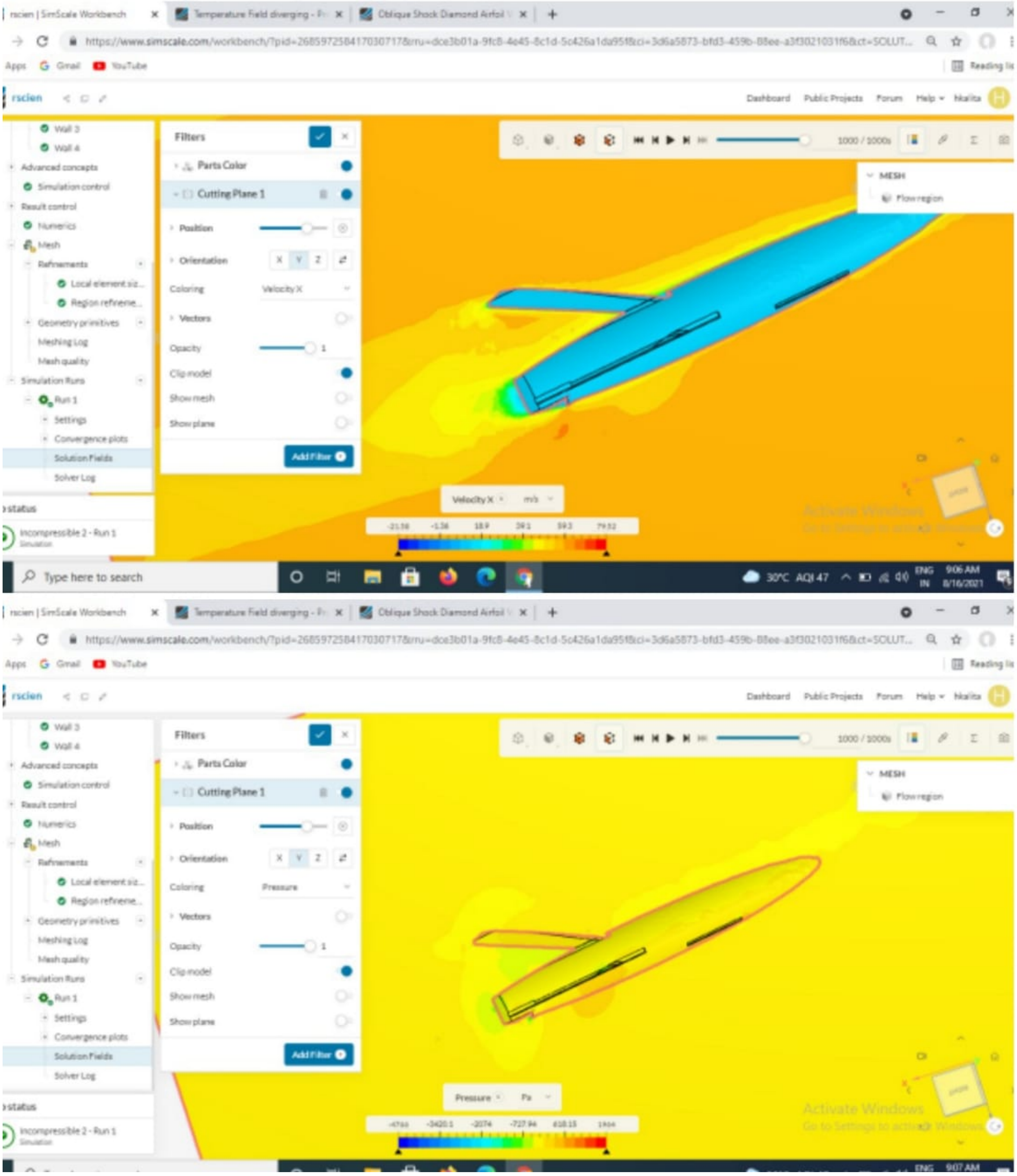


Figure 23: Caption

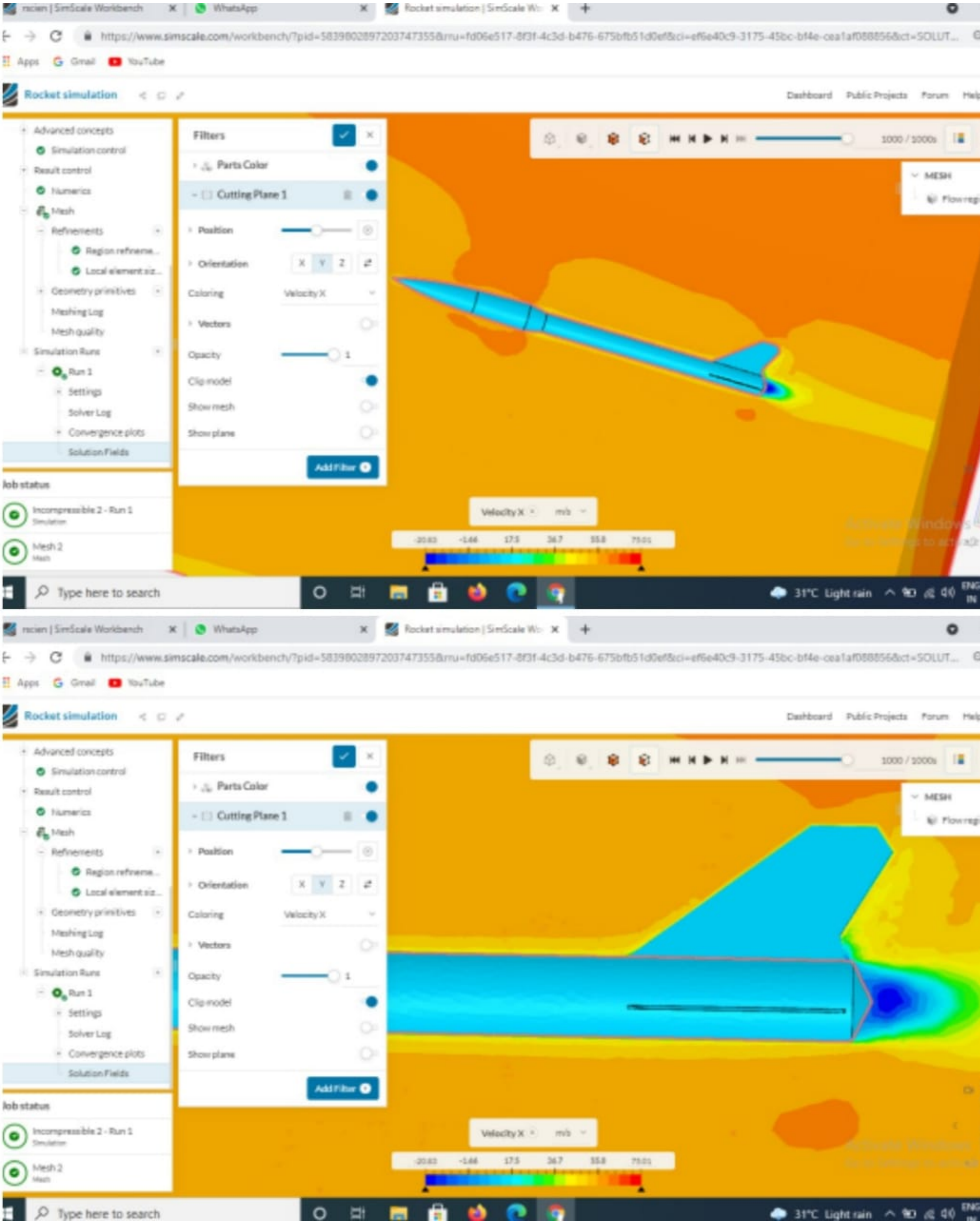


Figure 24: improved rocket

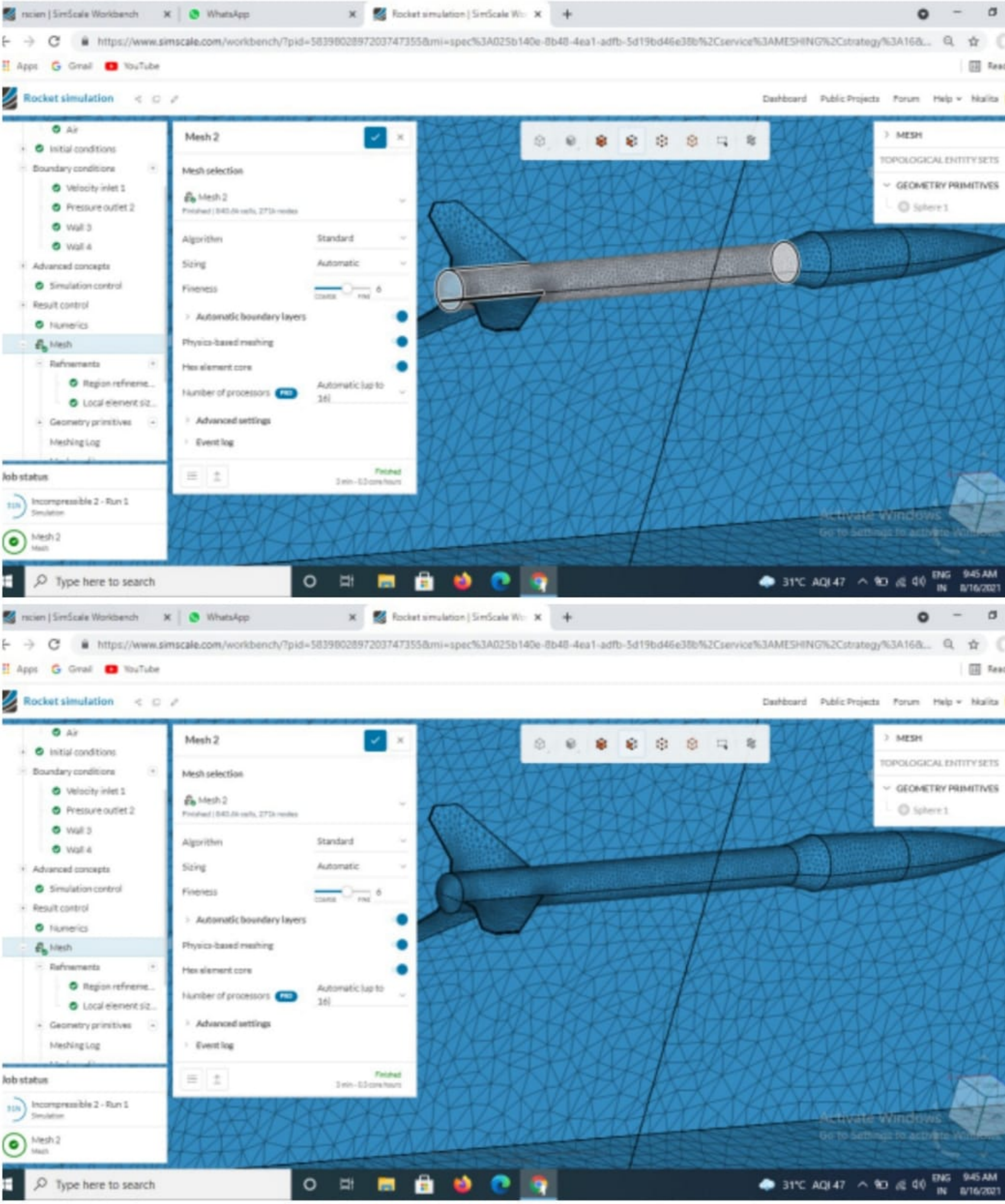


Figure 25: mesh of the modified rocket

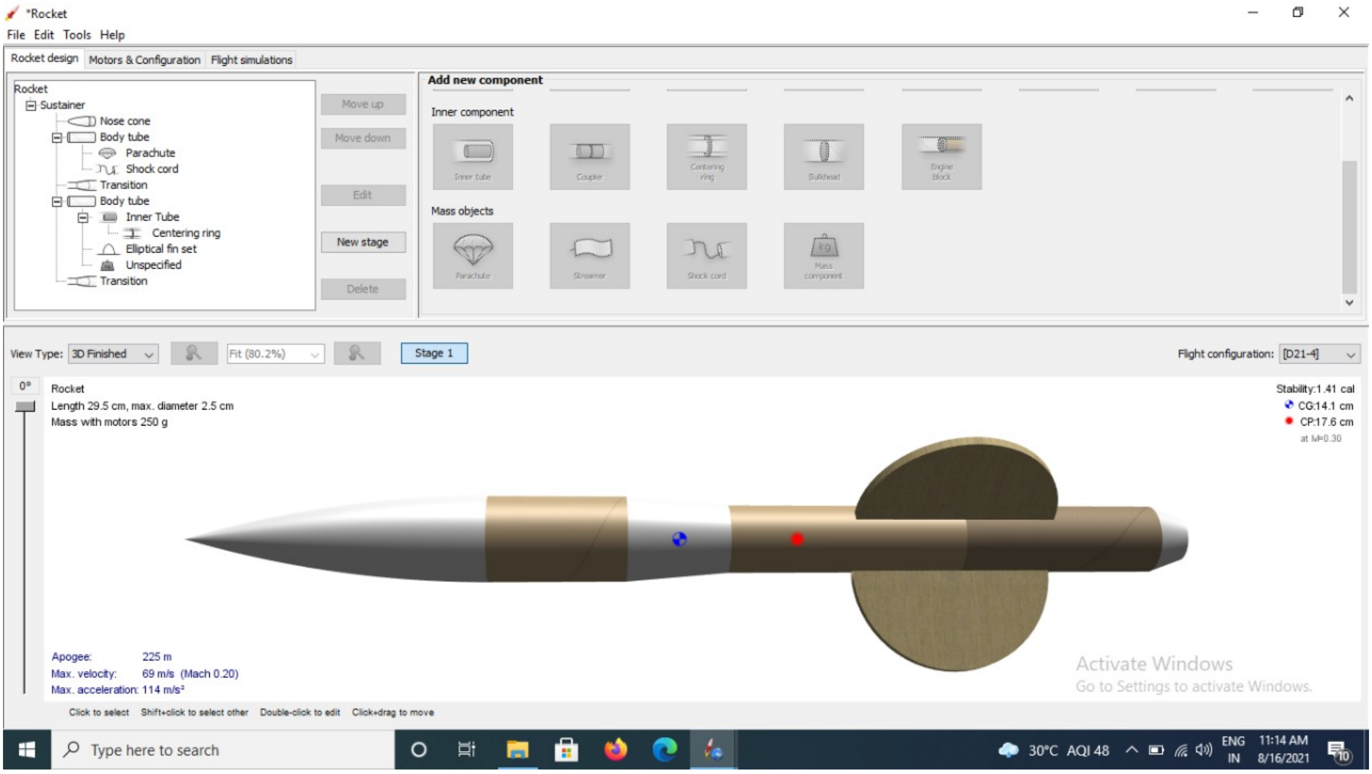


Figure 26: final rocket

After completing all the mechanical models,body ,adjusting different parts of the rocket and completing all the connections in the PCB we now about to come to the end of the project.

Just one thing left,i.e to upload the final code for the flight computer——
For the flight computer the **arduino code** is given below

```
#include <SD.h>
#include <Wire.h>
#include <SFE_BMP180.h>
#include<Servo.h>
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif
#define INTERRUPT_PIN 2  //I2C Interrupt Pin for MPU6050, use pin 2 on Arduino Uno & most boards
#define BUZZER_PIN 3     //set buzzer pin
#define BLUE_LED_PIN 8   //set blue LED pin
#define YELLOW_LED_PIN 9 //set yellow LED pin
#define RED_LED_PIN 10   //set red LED pin
#define BUTTON_PIN 7     //set button pin
#define chipSelect 4     //set chip select pin for MicroSD Card Adapter (CS pin)
```

```
SFE_BMP180 pressure;

Servo servo;

int Check_downs = 0;

double baseline;
//declare general use variables
int buttonState = 0;
int MODE = 0;         //initialize mode to zero
int t = 0;            //create timestamp value
int dataRate = 10;   //set specified sampling rate (data points per second)
     (somewhere between 10-200 is ideal)

//declare MPU control/status vars
bool blinkState = false;
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 =
    success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

//declare orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor
    measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor
    measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container
    and gravity vector

//create objects
File file;
SFE_BMP180 BMP;
MPU6050 mpu;

//Interrupt Detection Routine
volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt
     pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}

void setup() {
  //--- Serial Debugging ---
  Serial.begin(9600);
  //--- Establish Pin Modes and turn off all LEDs ---
```

```
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(BLUE_LED_PIN, OUTPUT);
  pinMode(YELLOW_LED_PIN, OUTPUT);
  pinMode(RED_LED_PIN, OUTPUT);
  pinMode(chipSelect, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
  pinMode(INTERRUPT_PIN, INPUT);
  digitalWrite(YELLOW_LED_PIN, LOW);
  digitalWrite(RED_LED_PIN, LOW);
  digitalWrite(BLUE_LED_PIN, LOW);
  tone(BUZZER_PIN, 500, 250);
  //initialize SD Card
  if(!SD.begin(chipSelect)){
    //Serial debugging
    Serial.println("Could not initialize SD card");
  }
  //clear SD data
  if(SD.exists("file.txt")){
    if(SD.remove("file.txt") == true){
      Serial.println("removed data");
    }
  }
  //initialize BMP sensor
  if(BMP.begin()){
    Serial.println("BMP init success");
  }
  //initialize IMU and I2C clock
  #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
      Wire.begin();
      Wire.setClock(400000);
  #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
      Fastwire::setup(400, true);
  #endif
  mpu.initialize(); //start MPU
  Serial.println(F("Testing device connections...")); //debugging serial
   statement
  Serial.println(mpu.testConnection() ? F("MPU6050 connection successful")
    : F("MPU6050 connection failed")); //debugging serial statement
  devStatus = mpu.dmpInitialize();
  // supply your own gyro offsets here, scaled for min sensitivity
  mpu.setXGyroOffset(0);
  mpu.setYGyroOffset(0);
  mpu.setZGyroOffset(0);
  mpu.setZAccelOffset(1688); // 1688 factory default for my test chip
  if (devStatus == 0) {
      // turn on the DMP, now that it's ready
      Serial.println(F("Enabling DMP..."));
      mpu.setDMPEnabled(true);
      // enable Arduino interrupt detection
      Serial.print(F("Enabling interrupt detection (Arduino external
   interrupt "));
      Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
```

```
      Serial.println(F(")..."));
      attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady,
    RISING);
      mpuIntStatus = mpu.getIntStatus();
      // set our DMP Ready flag so the main loop() function knows it's
    okay to use it
      Serial.println(F("DMP ready! Waiting for first interrupt..."));
      dmpReady = true;
      // get expected DMP packet size for later comparison
      packetSize = mpu.dmpGetFIFOPacketSize();
  } else {

      Serial.print(F("DMP Initialization failed (code "));
      Serial.print(devStatus);
      Serial.println(F(")"));
  }
  servo.attach(9);
  //set mode
  MODE = 1; //set to PAD IDLE mode - initialize sensors and SD card
  //MODE = 2; //set to FLIGHT mode - log data
  //MODE = 3; //set to RECOVERY mode - close file
}

void loop() {
  servo.write(90);
  delay(100);
  // put your main code here, to run repeatedly:
  if(MODE == 1){   //PAD IDLE MODE
    digitalWrite(RED_LED_PIN, HIGH);
    file = SD.open("file.txt", FILE_WRITE); //Open SD card file
    if(file) {
      Serial.println("t,T,P,ax,ay,az,rx,ry,rz"); //print first line with
    data labels
      file.println("t,T,P,ax,ay,az,rx,ry,rz");
      MODE = 2;
    }
    else{
      Serial.println("Error opening file");
      delay(5000); //just chill for 5 seconds before trying again
    }
  }
  if(MODE == 2){ //ACTIVE FLIGHT mode
    digitalWrite(YELLOW_LED_PIN, HIGH);
    digitalWrite(RED_LED_PIN, LOW);
    digitalWrite(BLUE_LED_PIN, LOW);
    //print timestamp and comma to separate data
    Serial.print(t);
    Serial.print(",");
    file.print(t);
    file.print(",");

    char status;
```

```
    double a,P,a1,a2,P2,Difference;


  P = getPressure();

  a1 = pressure.altitude(P,baseline);

  delay(3000);

  P2 = getPressure();

  a2 = pressure.altitude(P2,baseline);

  Difference = a2 - a1;
  if (abs(Difference) > 1) {
    if (Difference < 0 ) Check_downs =+ 1;
    if (Difference > 0 ) Check_downs = 0;
  }

  Serial.print("Start >");
  Serial.print(a1);
  Serial.print(" meter >");
  Serial.println(a2);

  Serial.print("Difference: ");
  Serial.print(a1 - a2);
  Serial.println("meters");
  Serial.print("Check downs : ");
  Serial.print(Check_downs);
  Serial.println();

  if (Check_downs == 3) {
    servo.write(180);
     digitalWrite(RED_LED_PIN ,LOW);
  digitalWrite(BLUE_LED_PIN,HIGH);

  }
  else
  { digitalWrite(RED_LED_PIN ,HIGH);
  digitalWrite(BLUE_LED_PIN,LOW);
  }
  delay(1000);

    file.print(",");
    //get IMU data
    if (!dmpReady) return;
    while (!mpuInterrupt && fifoCount < packetSize){
      if (mpuInterrupt && fifoCount < packetSize){
        fifoCount = mpu.getFIFOCount();
      }
    }
    mpuInterrupt = false;
```

```
    mpuIntStatus = mpu.getIntStatus();
    fifoCount = mpu.getFIFOCount();
    if((mpuIntStatus & _BV(MPU6050_INTERRUPT_FIFO_OFLOW_BIT)) || fifoCount
    >=1024){
      mpu.resetFIFO();
      fifoCount = mpu.getFIFOCount();
      Serial.println("FIFO Overflow!");
    }
    else if(mpuIntStatus & _BV(MPU6050_INTERRUPT_DMP_INT_BIT)){
      while(fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
      mpu.getFIFOBytes(fifoBuffer, packetSize);
      fifoCount -= packetSize;
      //get real-world acceleration
      mpu.dmpGetQuaternion(&q, fifoBuffer);
      mpu.dmpGetAccel(&aa, fifoBuffer);
      mpu.dmpGetGravity(&gravity, &q);
      mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
      //print real-world acceleration
      Serial.print(aaReal.x);
      file.print(aaReal.x);
      Serial.print(",");
      file.print(",");
      Serial.print(aaReal.y);
      file.print(aaReal.y);
      Serial.print(",");
      file.print(",");
      Serial.print(aaReal.z);
      file.print(aaReal.z);
      Serial.print(",");
      file.print(",");
      //get Euler angles
      mpu.dmpGetQuaternion(&q, fifoBuffer);
      mpu.dmpGetEuler(euler, &q);
      //print Euler angles
      Serial.print(euler[0]*180/M_PI);
      file.print(euler[0]*180/M_PI);
      Serial.print(",");
      file.print(",");
      Serial.print(euler[1]*180/M_PI);
      file.print(euler[1]*180/M_PI);
      Serial.print(",");
      file.print(",");
      Serial.print(euler[2]*180/M_PI);
      file.print(euler[2]*180/M_PI);
    }
    //end data entry line
    Serial.println(); //ends line
    file.println(); //ends line
    //check for mode switch
    buttonState = digitalRead(BUTTON_PIN);
    if(buttonState == LOW){
      MODE = 3;
```

```
      tone(BUZZER_PIN, 1000, 250);
      delay(100);
    }
  }
  if(MODE == 3){ //RECOVERY MODE
    file.close();
    digitalWrite(YELLOW_LED_PIN, LOW);
    digitalWrite(RED_LED_PIN, LOW);
    digitalWrite(BLUE_LED_PIN, HIGH);
    delay(1000);
  }
  t = t + 1;                //increment t value
  delay(1000/dataRate);     //pause so that data output corresponds to data rate
  if(t > 32765){            //prevents issues related to integers rolling over at 32767
    t = 1;
  }
}
double getPressure()
{
  char status;
  double T,P,p0,a;


  status = pressure.startTemperature();
  if (status != 0)
  {
    // Wait for the measurement to complete:

    delay(status);


    status = pressure.getTemperature(T);
    if (status != 0)
    {

      status = pressure.startPressure(3);
      if (status != 0)
      {
        // Wait for the measurement to complete:
        delay(status);



        status = pressure.getPressure(P,T);
        if (status != 0)
        {
          return(P);
        }
        else Serial.println("error retrieving pressure measurement\n");
      }
```

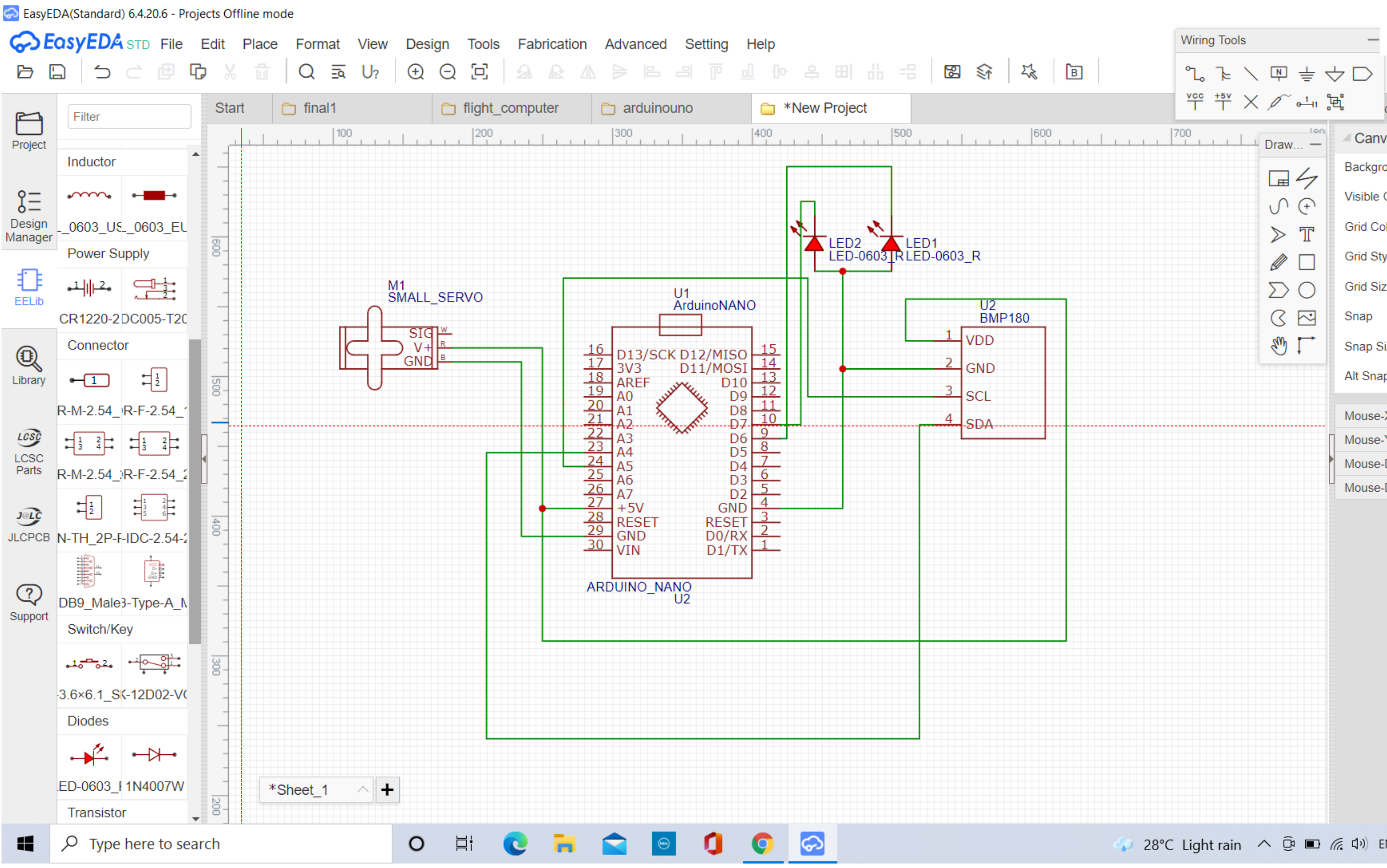


Figure 27: Basic flight computer

```
        else Serial.println("error starting pressure measurement\n");
      }
      else Serial.println("error retrieving temperature measurement\n");
    }
    else Serial.println("error starting temperature measurement\n");
  }

```

But the flight computer can me made with out MPU6050 and SD card.The circuit diagram is given in figure 28. **Here is the final code of the alternative /easy fight computer:**

```
 #include <SFE_BMP180.h>
#include <Wire.h>
#include <Servo.h>

// You will need to create an SFE_BMP180 object, here called "pressure":

SFE_BMP180 pressure;
```

```
Servo servo;
int GREENLED=9;
int REDLED=10;
int Check_downs = 0;

double baseline; // baseline pressure

void setup()
{
  Serial.begin(9600);
  Serial.println("REBOOT");

  // Initialize the sensor (it is important to get calibration values
    stored on the device).
  pinMode(REDLED,OUTPUT);
  pinMode(GREENLED,OUTPUT);
  if (pressure.begin()){
    Serial.println("BMP180 init success");
  }
  else
  {

    Serial.println("BMP180 init fail (disconnected?)\n\n");
    while(1); // Pause forever.
  }

  // Get the baseline pressure:

  baseline = getPressure();

  Serial.print("baseline pressure: ");
  Serial.print(baseline);
  Serial.println(" mb");
  Serial.print("baseline pressure: ");
  Serial.print(baseline/33.864,0);
  Serial.println(" Inhg");

  servo.attach(5);
  servo.write(90); //sets servo to its midpoint
}

void loop()
{servo.write(90);
delay(10);
  double a,P,a1,a2,P2,Difference;

  // Get a new pressure reading:

  // Show the relative altitude difference between
  // the new reading and the baseline reading:

  //a = pressure.altitude(P,baseline);
```

```
  //Serial.print("relative altitude: ");
  //if (a >= 0.0) Serial.print(" "); // add a space for positive numbers
  //Serial.print(a,1);
  //Serial.print(" meters, ");
  //if (a >= 0.0) Serial.print(" "); // add a space for positive numbers

  P = getPressure();

  a1 = pressure.altitude(P,baseline);

  delay(3000);

  P2 = getPressure();

  a2 = pressure.altitude(P2,baseline);

  Difference = a2 - a1;
  if (abs(Difference) > 1) {
    if (Difference < 0 ) Check_downs =+ 1;
    if (Difference > 0 ) Check_downs = 0;
  }

  Serial.print("Start >");
  Serial.print(a1);
  Serial.print(" meter >");
  Serial.println(a2);

  Serial.print("Difference: ");
  Serial.print(a1 - a2);
  Serial.println("meters");
  Serial.print("Check downs : ");
  Serial.print(Check_downs);
  Serial.println();

  if (Check_downs == 3) {
    servo.write(90);
      digitalWrite(REDLED,LOW);
  digitalWrite(GREENLED,HIGH);

  }
  else
  { digitalWrite(REDLED,HIGH);
  digitalWrite(GREENLED,LOW);
  }
  delay(1000);

}


double getPressure()
{
```

```
  char status;
  double T,P,p0,a;


  status = pressure.startTemperature();
  if (status != 0)
  {
    // Wait for the measurement to complete:

    delay(status);


    status = pressure.getTemperature(T);
    if (status != 0)
    {

      status = pressure.startPressure(3);
      if (status != 0)
      {
        // Wait for the measurement to complete:
        delay(status);

        // Retrieve the completed pressure measurement:
        // Note that the measurement is stored in the variable P.
        // Use '&P' to provide the address of P.
        // Note also that the function requires the previous temperature
 measurement (T).
        // (If temperature is stable, you can do one temperature
 measurement for a number of pressure measurements.)
        // Function returns 1 if successful, 0 if failure.

        status = pressure.getPressure(P,T);
        if (status != 0)
        {
          return(P);
        }
        else Serial.println("error retrieving pressure measurement\n");
      }
      else Serial.println("error starting pressure measurement\n");
    }
    else Serial.println("error retrieving temperature measurement\n");
  }
  else Serial.println("error starting temperature measurement\n");
}
```