UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

Design and Construction of a Vertical Landing Vehicle using a Cold Gas Thruster

Document:

Annexes

Author:

Sylvain L. Walsh

Director /Co-director:

Jaume Solé Bosquet

Degree:

Bachelor in Aerospace Vehicle Engineering

Examination session:

Spring, 2021

BACHELOR FINAL THESIS

ESEIAAT
Final Bachelor Thesis

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

UNIVERSITAT POLITÈCNICA DE CATALUNYA

BACHELOR IN AEROSPACE VEHICLE ENGINEERING

Design and Construction of a Vertical Landing Vehicle using a Cold Gas Thruster

ANNEXES

Author:
Sylvain L. Walsh

Director:
Jaume Solé Bosquet

June 22, 2021

Contents

A

Matlab Algorithm Code

This annex includes the codes of the algorithms developed in Matlab

A.1 Required Thrust Algorithm

```
clear
clc
close all

%varible definition
m=[1.5]; %mass
rho=1.225; %air density @SL
g=9.81; %gravitational acceleration
h_d=[20]; %drop height
h_a=0.95; %aerobrake deployment height factor
h_i=0.3; %ignition height factor
dt=0.001;
dT=0.0001;
Cda=1.4242;
Cdr=0.89136;
v0e=0.1;
h0e=0.1;

%initial calculations
W=m*g;
Srr=pi/4*0.1^2; %ref surface rocket
Sar=0.0163*4; %ref surface area aerobrakes

for i=1:numel(h_d)
    for j=1:numel(m)

        treq(i,j)=m(j)*g;

    end

end

%main

for i=1:numel(h_d)

    for j=1:numel(W)
        touchdown=false;
        o=0;
        while touchdown==false

            o=o+1;
            terminate=false;
            contact=false;
            vneg=false;
            v0=false;
```

```
47          k=1;
48          v(i,j,k)=0;
49          a(i,j,k)=g;
50          h(i,j,k)=h_d(i);
51          Dr(i,j,k)=0;
52          Da(i,j,k)=0;
53          a_t(i,j,k)=0;
54          a_d(i,j,k)=0;
55          t=dt;
56
57          while terminate==false
58
59              h_per=h(i,j,k)/h_d(i);
60
61              Dr(i,j,k)=0.5*rho*Srr*Cdr*v(i,j,k)^2;
62
63              if h_per ≤h_a
64
65                  Da(i,j,k)=0.5*rho*Sar*Cda*v(i,j,k)^2;
66              else
67                  Da(i,j,k)=0;
68              end
69
70              if h_per ≤h_i
71
72                  a_t(i,j,k)=treq(i,j)/m(j);
73
74              else
75
76                  a_t(i,j,k)=0;
77
78              end
79
80              D(i,j,k)=Da(i,j,k)+Dr(i,j,k);
81              a_d(i,j,k)=D(i,j,k)/m(j);
82
83              a(i,j,k)=g-a_d(i,j,k)-a_t(i,j,k);
84
85              v(i,j,k+1)=v(i,j,k)+a(i,j,k)*dt;
86
87              h(i,j,k+1)=h(i,j,k)-(v(i,j,k)*dt+0.5*a(i,j,k)*dt^2);
88
89              if v(i,j,k+1)> -v0e && v(i,j,k)< v0e && h_per<0.5
90
91                  v0=true;
92
93              end
94
95              if v(i,j,k+1)<-2*v0e
96
97                  vneg=true;
98                  terminate=true;
99
100             end
101
102             if h(i,j,k+1)<h0e
103
104                 contact=true;
105                 terminate=true;
106
107             end
108
109
110             t=t+dt;
111             k=k+1;
112         end
113
```

```
114             if contact==true && v0==true
115
116                 touchdown=true;
117
118             else
119                 if vneg==true
120
121                     treq(i,j)=treq(i,j)-(dT/2);
122
123
124                 else
125
126                     treq(i,j)=treq(i,j)+dT;
127
128                 end
129
130             end
131
132
133         end
134         a(i,j,k)=a(i,j,k-1);
135         t_f(i,j)=(k-1)*dt;
136     end
137
138 end
139 set(groot,'defaultLineLineWidth',2.0)
140 for i=1:numel(h_d)
141     txt=['Drop height=',num2str(h_d(i)),'m'];
142     figure('Name', txt)
143
144     for j=1:numel(m)
145         clear v_plot a_plot h_plot t_vec
146         t_vec=0:dt:t_f(i,j);
147         for l=1:numel(t_vec)
148
149             v_plot(l)=v(i,j,l);
150             a_plot(l)=a(i,j,l);
151             h_plot(l)=h(i,j,l);
152
153         end
154         subplot(2,2,j)
155         plot(t_vec,v_plot)
156         hold on
157         plot(t_vec,a_plot)
158         hold on
159         plot(t_vec,h_plot)
160         legend('Velocity (m/s)', 'Acceleration (m/s^2)', 'Height (m)')
161         grid on
162         xlabel('time (s)')
163         txt=['Velocity, acceleration and height for a mass of ', num2str(m(j)),'kg. ...
    Required thrust = ', num2str(treq(i,j)),'N'];
164         title(txt)
165         xlim([0,t_f(i,j)])
166     end
167 end
168 figure('Name','Thrust-weight')
169 for i=1:numel(h_d)
170     txt=['Drop height=',num2str(h_d(i)),'m'];
171     plot(m,treq(i,:),'Displayname',txt)
172     legend
173     xlabel('mass (kg)')
174     ylabel('Required thrust (N)')
175     grid on
176     hold on
177
178 end
```

A.2 Burn Algorithm

```
clear
clc
close all
%variable decalration
Pe=101325; %Pa
Te=293; %K
Pt_i=206.8427*1e5;
Pr=20.68*1e5;%Pa
Pc=Pr;
Tc=Te; %Negligating Joule Thomson effect
V=[0.21]*1e-3; %Tank volume | Liters to m^3;
Th=[20]; %Thrust in Newtons
alpha=deg2rad(15);
g=9.81;
dt=0.0001;
Rsp=287.058; %Specific gas constant for dry air J/kg*k
gamma=1.4;

%initial calgulations
lambda=(1+cos(alpha))/2;



%main
for i=1:numel(V)

    for j=1:numel(Th)

        k=1;
        t=0;
        %initial conditions
        Pt(i,j,k)=Pt_i;
        [Z(i,j,k)]=compress_factor(Pt(i,j,k), Tc);
        rho(i,j,k)=Pt(i,j,k)/(Rsp*Tc*Z(i,j,k));
        m(i,j,k)=rho(i,j,k)*V(i);
        Isp(i,j,k)=(1/g)*sqrt((2*gamma/(gamma-1))*Rsp*Tc*(1-(Pe/Pc)^((gamma-1)/gamma)));
        mdot(i,j,k)=Th(j)/(g*lambda*Isp(i,j,k));



        while Pt(i,j,k)>Pe+10000

            k=k+1;
            t=t+dt;
            m(i,j,k)=m(i,j,k-1)-mdot(i,j,k-1)*dt;
            rho(i,j,k)=m(i,j,k)/V(i);
            [Z(i,j,k)]=compress_factor(Pt(i,j,k-1), Tc);
            Pt(i,j,k)=Z(i,j,k)*rho(i,j,k)*Tc*Rsp;
            Isp(i,j,k)=(1/g)*sqrt((2*gamma/(gamma-1))*Rsp*Tc*(1-(Pe/Pc)^((gamma-1)/gamma)));
            mdot(i,j,k)=Th(j)/(g*lambda*Isp(i,j,k));
            %Q(i,j,k)=60*1000*mdot(i,j,k)/rho(i,j,k);
            Cv(i,j,k)=0.865*(mdot(i,j,k)*3600)/(27.3* 2/3 * sqrt(11.82*Pc*1e-5));

        end
        t_f(i,j)=(k-1)*dt;

    end

end

for i=1:numel(V)
    txt=['Burn parameters for =',num2str(V(i)*1000),'L tank'];
    figure('Name', txt)
```

```
64      for j=1:numel(Th)
65          clear m_plot rho_plot Pt_plot mdot_plot Isp_plot z_plot t_vec Q_plot
66          t_vec=0:dt:t_f(i,j);
67          for l=1:numel(t_vec)
68
69              m_plot(l)=m(i,j,l);
70              rho_plot(l)=rho(i,j,l);
71              Pt_plot(l)=Pt(i,j,l);
72              mdot_plot(l)=mdot(i,j,l);
73              Isp_plot(l)=Isp(i,j,l);
74              Z_plot(l)=Z(i,j,l);
75              %Q_plot(l)=Q(i,j,l);
76              Cv_plot(l)=Cv(i,j,l);
77
78          end
79          %subplot(2,2,j)
80           plot(t_vec,mdot_plot)
81           hold on
82           %plot(t_vec,rho_plot)
83           hold on
84           %plot(t_vec,Pt_plot/100000)
85           hold on
86           %plot(t_vec,Isp_plot)
87           hold on
88           plot(t_vec,Cv_plot)
89           plot(t_vec,m_plot);
90
91          legend('Mass flow rate mdot (kg/s) ', 'Flow coefincient K_v (m^3/h)', 'Propellant ...
    mass m (kg)')
92          %legend('Tank density \rho (kg/m^3)', 'Tank pressure P_c (Bar)', 'Isp (s)')
93          grid on
94          xlabel('time (s)')
95          txt=['Constant ', num2str(Th(j)),'N thrust'];
96          title(txt)
97          xlim([0.1,t_f(i,j)])
98      end
99
100 end
```

A.2.1 Z Compressibility Factor Function

```
function [Z] = compress_factor(P,T)
P=P*1e-5; %Pa to bar

z=[0.0052   0.026   0.0519  0.1036  0.2063  0.3082  0.4094  0.5099  0.7581  1.0125  NaN NaN ...
    NaN NaN
NaN 0.025   0.0499  0.0995  0.1981  0.2958  0.3927  0.4887  0.7258  0.9588  1.1931  1.4139  ...
    NaN NaN
0.9764  0.0236  0.0453  0.094   0.1866  0.2781  0.3686  0.4681  0.6779  0.8929  1.1098  ...
    1.311   1.7161  2.1105
0.9797  0.8872  0.0453  0.09    0.1782  0.2635  0.3498  0.4337  0.6386  0.8377  1.0395  ...
    1.2227  1.5937  1.9536
0.988   0.9373  0.886   0.673   0.1778  0.2557  0.3371  0.4132  0.5964  0.772   0.953   ...
    1.1076  1.5091  1.7366
0.9927  0.9614  0.9205  0.8297  0.5856  0.3313  0.3737  0.434   0.5909  0.7699  0.9114  ...
    1.0393  1.3202  1.5903
0.9951  0.9748  0.9489  0.8954  0.7803  0.6603  0.5696  0.5489  0.634   0.7564  0.884   ...
    1.0105  1.2585  1.497
0.9967  0.9832  0.966   0.9314  0.8625  0.7977  0.7432  0.7084  0.718   0.7986  0.9 1.0068  ...
    1.2232  1.4361
0.9978  0.9886  0.9767  0.9539  0.91    0.8701  0.8374  0.8142  0.8061  0.8549  0.9311  ...
    1.0185  1.2054  1.3944
0.9992  0.9957  0.9911  0.9822  0.9671  0.9549  0.9463  0.9411  0.945   0.9713  1.0152  ...
    1.0702  1.199   1.3392
0.9999  0.9987  0.9974  0.995   0.9917  0.9901  0.9903  0.993   1.0074  1.0326  1.0669  ...
    1.1089  1.2073  1.3163
1   1.0002  1.0004  1.0014  1.0038  1.0075  1.0121  1.0183  1.0377  1.0635  1.0947  1.1303  ...
    1.2116  1.3015
1.0002  1.0012  1.0025  1.0046  1.01    1.0159  1.0229  1.0312  1.0533  1.0795  1.1087  ...
    1.1411  1.2117  1.289
1.0003  1.0016  1.0034  1.0063  1.0133  1.021   1.0287  1.0374  1.0614  1.0913  1.1183  ...
    1.1463  1.209   1.2778
1.0003  1.002   1.0034  1.0074  1.0151  1.0234  1.0323  1.041   1.065   1.0913  1.1183  ...
    1.1463  1.2051  1.2667
1.0004  1.0022  1.0039  1.0081  1.0164  1.0253  1.034   1.0434  1.0678  1.092   1.1172  ...
    1.1427  1.1947  1.2475
1.0004  1.002   1.0038  1.0077  1.0157  1.024   1.0321  1.0408  1.0621  1.0844  1.1061  ...
    1.1283  1.172   1.215
1.0004  1.0018  1.0037  1.0068  1.0142  1.0215  1.029   1.0365  1.0556  1.0744  1.0948  ...
    1.1131  1.1515  1.1889];

z1=[1 5 10 20 40 60 80 100 150 200 250 300 400 500];
z2=[75 80 90 100 120 140 160 180 200 250 300 350 400 450 500 600 800 1000];



for m=1:size(z,1)
    if T≥z2(m) && T≤z2(m+1)
        py1=m;
        py3=m+1;
    end
end

px1=1;
px3=1;
for n=1:size(z,2)
    if P≥z1(n) && P≤z1(n+1)
        px1=n;
        px3=n+1;
    end
end

if px1==1 && px3==1

    x2_y1=z(py1,1);
```

```
47      x2_y3=z(py3,1);
48
49  else
50
51      x2_y1=z(py1,px1)+((z(py1,px3)-z(py1,px1))/(z1(px3)-z1(px1)))*(P-z1(px1));
52      x2_y3=z(py3,px1)+((z(py3,px3)-z(py3,px1))/(z1(px3)-z1(px1)))*(P-z1(px1));
53  end
54
55  Z=x2_y1+(((x2_y3-x2_y1)/(z2(py3)-z2(py1)))*(T-z2(py1)));
56  end
```

B

Arduino Program Code

This section contains the codes developed for the Arduino

B.1 Thrust Test Bench

```
#include <HX711_ADC.h>
#if defined(ESP8266)|| defined(ESP32) || defined(AVR)
#include <EEPROM.h>
#endif

//pins:
const int HX711_dout = 4; //mcu > HX711 dout pin
const int HX711_sck = 5; //mcu > HX711 sck pin
long Time_i;
char command;
bool burning;
float a;
int solenoid=2;

HX711_ADC LoadCell(HX711_dout, HX711_sck);

const int calVal_eepromAdress = 0;
unsigned long t = 0;

void setup() {
  pinMode(solenoid, OUTPUT);
  Serial.begin(9600);
  Serial.println();
  Serial.println("Starting...");

  LoadCell.begin();
  float calibrationValue; // calibration value
  EEPROM.get(calVal_eepromAdress, calibrationValue);

  unsigned long stabilizingtime = 2000;
  boolean _tare = true;
  LoadCell.start(stabilizingtime, _tare);
  if (LoadCell.getTareTimeoutFlag()) {
    Serial.println("Timeout, check MCU>HX711 wiring and pin designations");
    while (1);
  }
```

```
37
38    else {
39      LoadCell.setCalFactor(calibrationValue); // set calibration value (float)
40      Serial.println("Startup is complete");
41    }
42    Serial.println();
43    Serial.println("Enter 'i' to initiate the burn. When burn is finilized hit 'e' to ...
        end");
44    bool ignition = false;
45    while (ignition == false) {
46      if (Serial.available() > 0) {
47        command = Serial.read();
48        if (command == 'i') {
49          int countdown = 5;
50
51          while (countdown != -1) {
52            Serial.print("T- ");
53            Serial.print(countdown);
54            Serial.println('s');
55            delay(1000);
56            countdown--;
57          }
58          Serial.println(" gnition ...") ;
59          ignition = true;
60        } else {
61          Serial.println("Error...");
62
63        }
64      }
65    }
66    long c = millis();
67    Time_i = -c;
68    burning = true;
69    digitalWrite(solenoid, HIGH);
70  }
71
72  void loop() {
73    while (burning == true) {
74      static boolean newDataReady = 0;
75      float Time = Time_i;
76
77      // check for new data/start next conversion:
78      if (LoadCell.update()) newDataReady = true;
79      float i = LoadCell.getData();
80      a = millis();
81      Time = Time + a;
82      Serial.print(Time / 1000);
83      Serial.print(" ");
84      Serial.println(i);
85      newDataReady = 0;
86
87
88
```

```
    // receive command from serial terminal, send 't' to initiate tare operation:
    if (Serial.available() > 0) {
      char inByte = Serial.read();
      if (inByte == 't') LoadCell.tareNoDelay();
      if (inByte == 'e') {
        burning = false;
        digitalWrite(solenoid, LOW);
        Serial.print("Brun complete");
      }
    }

    // check if last tare operation is complete:
    if (LoadCell.getTareStatus() == true) {
      Serial.println("Tare complete");
    }
  }
}
```

B.2 Flight Computer Program

```
#include <TFLI2CAlt.h>
#include <Servo.h>
#include <JY901.h>

#define DEG2RAD(x) (x * 3.141592 / 180)

#define SOLENOID 4
#define SAFTEY A0
#define COM_LED A1
#define AIR_BRAKES 12
#define LANDING_LEGS 11

#define SET_POINT 0
float kp = 0.2436;
//float ki = 0.21079;
float kd = 0.3418;
float current_time, previous_time, dt, error[2], last_error[2], cumulative_error[2], ...
    derivative_error[2], output[2];
int j;
unsigned long air_delay;
unsigned long leg_delay;
bool air_deploy = false;
bool leg_deploy = false;
int heat_time = 500;

#define IGNITION HIGH
#define BURN_OUT LOW

#define X_OFFSET -91.85
#define Y_OFFSET 0.96
#define LEG_DEP_HEIGHT 10
```

```
31
32 #define X 0
33 #define Y 1
34 #define Z 2
35 #define SERVO_X 5
36 #define SERVO_Y 6
37 #define SERVO_LIM 15
38 float SERVO_M[2] = {3.33, -3.33};
39 int SERVO_N[2] = {85, 110};
40
41 Servo servos[2];
42
43 void setup() {
44   JY901.StartIIC();
45   Serial.begin(9600);
46   // Initialize pins
47   pinMode(SOLENOID, OUTPUT);
48   pinMode(SAFTEY, INPUT);
49   pinMode(AIR_BRAKES, OUTPUT);
50   pinMode(LANDING_LEGS, OUTPUT);
51   pinMode(COM_LED, OUTPUT);
52   servos[X].attach(SERVO_X);
53   servos[Y].attach(SERVO_Y);
54   setServo(X, 0);
55   setServo(Y, 0);
56   setSolenoid(BURN_OUT);
57   digitalWrite(AIR_BRAKES, LOW);
58   digitalWrite(LANDING_LEGS, LOW);
59   digitalWrite(COM_LED, HIGH);
60   allGo();
61   initDrop();
62   setSolenoid(IGNITION);
63   digitalWrite(AIR_BRAKES, HIGH);
64   air_delay = millis() + heat_time;
65
66 }
67
68 void loop() {
69   if (j == 20 && leg_deploy == false) {
70     refreshLidar();
71     j = 0;
72   }
73   j++;
74   heatShut();
75   refreshIMU();
76   PID(getAngles());
77   setServo(X, output[X]);
78   setServo(Y, output[Y]);
79 }
80
81 void setSolenoid (int state) {
82   digitalWrite(SOLENOID, state);
83 }
```

```
84
85 void setServo(int servo, int posPID) {
86
87   int servo_pos = SERVO_M[servo] * posPID + SERVO_N[servo];
88   servos[servo].write(servo_pos);
89 }
90
91 void allGo() {
92   unsigned long led_time = millis() + 500;
93   int A;
94   int B = 0;
95   while (true) {
96     if (millis() ≥ led_time) {
97       A = commute(B);
98       B = A;
99       led_time += 500;
100    }
101    if (digitalRead(SAFTEY) == HIGH) {
102      break;
103    }
104  }
105 }
106
107 void initDrop () {
108   unsigned long led_time_ = millis() + 150;
109   int A_;
110   int B_ = 0;
111   while (true) {
112     refreshIMU();
113     if (millis() ≥ led_time_) {
114       A_ = commute(B_);
115       B_ = A_;
116       led_time_ += 150;
117     }
118     if (getAcc(Z) < 0.7) {
119       break;
120     }
121   }
122   digitalWrite(COM_LED, HIGH);
123 }
124
125 int PID(float input[2]) {
126   current_time = millis();
127   dt = current_time - previous_time;
128
129   error[X] = SET_POINT - input[X];
130   error[Y] = SET_POINT - input[Y];
131   cumulative_error[X] += error[X] * dt;
132   cumulative_error[Y] += error[Y] * dt;
133   derivative_error[X] = (error[X] - last_error[X]) / dt;
134   derivative_error[Y] = (error[Y] - last_error[Y]) / dt;
135
136   output[X] = kp * error[X] + /*ki * cumulative_error[X] +*/ kd * derivative_error[X];
```

```
  output[Y] = kp * error[Y]  + /*ki * cumulative_error[Y] +*/ kd * derivative_error[Y];

  for (int i = 0; i < 2; i++) {
    if (output[i] > SERVO_LIM) {
      output[i] = SERVO_LIM;
    }
    if (output[i] < -SERVO_LIM) {
      output[i] = - SERVO_LIM;
    }
  }

  last_error[X] = error[X];
  last_error[Y] = error[Y];
  previous_time = current_time;
}

int commute(int val) {
  if (val == 1) {
    val = 0;
    digitalWrite(COM_LED, LOW);
  } else {
    val = 1;
    digitalWrite(COM_LED, HIGH);
  }

  return val;
}

void heatShut() {
  if (air_deploy == true) {
    if (millis() ≥ air_delay) {
      digitalWrite(AIR_BRAKES, LOW);
    }
  }

  if (leg_deploy == true) {
    if (millis() ≥ leg_delay) {
      digitalWrite(LANDING_LEGS, LOW);
    }
  }
}

int16_t LIDAR_dist  = 0;
int16_t bin;

double true_dist = 0;

const int HEADER = 0x59;
unsigned char uart[9];
int lidar_check;

TFLI2CAlt tfli2c;

```

```
190 void refreshLidar() {
191   tflI2C.getData(LIDAR_dist, bin, bin, TFL_DEFAULT_ADDR);
192   ////  true_dist = LIDAR_dist;
193   true_dist = sqrt(pow(double(LIDAR_dist), 2) / (pow(tan(DEG2RAD(getAngle(X))), 2) + ...
        pow(tan(DEG2RAD(getAngle(Y))), 2) + 1));
194 }
195
196 int getAltitude() {
197   if (true_dist == 0) {
198     true_dist = 1000;
199   } else if (true_dist ≤ LEG_DEP_HEIGHT) {
200     leg_deploy == true;
201     digitalWrite(LANDING_LEGS, HIGH);
202     leg_delay = millis() + heat_time;
203   }
204   return (true_dist);
205 }
206
207 float imu_angle[2];
208 float imu_acc_z;
209
210 void refreshIMU () {
211   JY901.GetAcc();
212   //  JY901.GetGyro();
213   JY901.GetAngle();
214   imu_angle[X] = X_OFFSET + (float)JY901.stcAngle.Angle[X] / 32768 * 180;
215   imu_angle[Y] = Y_OFFSET + (float)JY901.stcAngle.Angle[Y] / 32768 * 180;
216   imu_angle[Z] = (float)JY901.stcAngle.Angle[Z] / 32768 * 180;
217   imu_acc_z = (float)JY901.stcAcc.a[Z] / 32768 * 16;
218 }
219
220 float getAcc (int axis) {
221   return (imu_acc_z);
222 }
223
224
225 float getAngle (int axis) {
226   return (imu_angle[axis]);
227 }
228
229 float* getAngles () {
230   return (imu_angle);
231 }
```

C

Mass Breakdown

Table C.1: Vehicle mass breakdown

		Mass [g]	QTY	Subtotal [g]
Propulsion system				**748**
	Tank 0.21L	335	1	335
	Regulator	175	1	175
	Stop valve	62	1	62
	Solenoid	176	1	176
Aerobrakes				**153**
	Fin holder	15	4	60
	Aerobrake	15	4	60
	Fin -body attachment	33	1	33
Landing legs				**112**
	Leg	11	4	44
	Strut	4	4	16
	Vehicle base	52	1	52
TVC				**20**
	Engine mount (nozzle)	5	1	5
	Inner gimbal	15	1	15
Electronics				**156**
	Arduino	40	1	40
	Battery	32	2	64
	Board, cables and components	52	1	52
Internal structure				**84**
	Solenoid holder	9	1	9
	Electronics and battery holder	22	1	22
	Tank holders	45	1	45
	IMU holder	8	1	8
External structure + miscellaneous				**287**
TOTAL [g]				**1563**