Politechnika Wrocławska

Symulacja apogeum lotu rakiety

- Dokonywana w czasie rzeczywistym, wersja 0

PWr in Space Mission Simulator

Krzysztof Gliwiński

Politechnika Wrocławska
Data wydania: 6 października 2021

PWR IN SPACE

Spis treści

1 Wstęp

Podczas zawodów rakiet eksperymentalnych duża część punktów bierze się z uzyskania odpowiedniego apogeum lotu, w zależności od konkurencji. W przypadku rakiety R4 jest to wysokość 10000 stóp czyli 3048 metrów.

W związku z tym że R4 leci na silniku hybrydowym, czas wypalenia utleniacza oraz zamknięcie zaworu między butlą a komorą spalania oznacza w istocie koniec działania silnika i generowania przez niego ciągu.

Jednym ze sposobów na dolecenie na idealną wysokość jest rzecz jasna zatankowanie odpowiedniej ilości utleniacza, co w praktyce okazuje się bardzo trudne - symulacje nie są wystarczająco dokładne oraz brakuje testów silnika, które i tak nie pozwalają na bezbłędne sprawdzenie ciągu.

Innym sposobem który był już prezentowany na zawodach jest zastosowanie hamulców powietrznych, które byłyby sterowane przez algorytm propabilistyczny lub sztuczną inteligencję.

Ostatnim, najprostszym i najsensowniejszym sposobem jest symulacja wysokości lotu dokonywana w czasie teraźniejszym przez komputer główny rakiety. Ze względu na brak możliwości zastosowania solvera równań różniczkowych, zaproponuję bardzo uproszczony sposób realizacji takiego algorytmu.

2 Zamysł oraz sposób działania

Moduł Pitota rakiety dostarcza wystarczająco dokładne dane na temat prędkości, żeby z tego skorzystać w trakcie lotu.

Zamysł jest bardzo prosty - posiadając tę informację komputer główny mógłby policzyć jak daleko doleci gdyby zamknął zawór w tym momencie. Jeśli symulacja wykaże wynik wyższy niż 3048 metrów - własnie to powinno się wydarzyć.

2.1 Równania ruchu

Zważając na brak żyroskopu w rakiecie który z dostateczną pewnością dałby nam odczyt nachylenia rakiety względem Ziemi, zakładam że rakieta leci pod kątem równym 90° co powinno być wartością bliską rzeczywistej jeśli rakieta się odpowiednio ustabilizuje.

W przypadku braku ciągu silnika, jedynymi siłami działającymi na rakietę są siła grawitacji - która ze względu na stałą masę rakiety po zamknięciu zaworu oraz niewysoki lot jest stała, oraz siła oporu aerodynamicznego, która zależna jest już od wielu czynników - prędkości rakiety, gęstości powietrza więc w istocie również wysokości itd.

Przy zastosowaniu zwykłych równań dynamiki Newtona opisywane zdarzenie można wyrazić takim układem:

$$dopisachere \tag{1}$$

Ze względu na wspomniany brak możliwości użycia solvera, proponuję użycie prostego algorytmu, który przyjmowałby siłę oporu jako stałą na pewien określony timestamp i iterował po kolejnych wartościach prędkości i wysokości.

2.2 Opis uproszczonej funkcji

W związku z tym, że siła oporu aerodynamicznego jest funkcją prędkości i wysokości $F_{op}(h, v)$, można na pewien krótki timestamp t_s przybliżyć ją funkcją stałą, o parametrach h, v w chwili t_s. Parametry te odczytywane są z modułu rurki Pitota.

Zatem zmiana drogi i prędkości w czasie $t_s = t_n - t_{n-1}$ z prostych równań ruchu wygląda następująco:

$$\begin{aligned} h(t_n) &= h(t_{n-1}) + v(t_{n-1}) \cdot t_s - \frac{gt_s^2}{2} - \frac{F_{op}(h,v) \cdot t_s^2}{2m_r} \\ v(t_n) &= v(t_{n-1}) - gt_s - \frac{F_{op}(h,v) \cdot t_s}{m_r} \end{aligned} \tag{2}$$

Gdzie m_r jest masą rakiety w momencie wyłączenia ciągu, g to przyspieszenie ziemskie równe w przybliżeniu na danych wysokościach $9,81$

Tutaj największym wyzwaniem jest wyliczenie siły oporu.

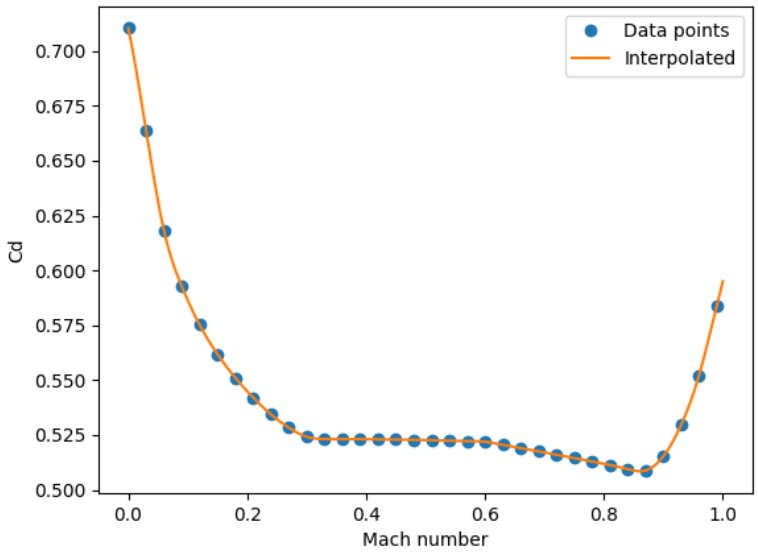
2.3 Obliczenia siły oporu

Siła oporu aerodynamicznego dana jest wzorem

$$\frac{C_d \ \rho v^2 A_{ref}}{2} \tag{3}$$

Gdzie C_d to współczynnik oporu aeryodynamicznego, ρ to gęstośc powietrza na wysokości h, v to prędkość rakiety a A_{ref} to pole powierzchni przekroju rakiety od strony czołowej. W tym przypadku to pole okręgu będącego korpusem oraz lotek.

Współczynnik C_d został obliczony dla rakiety R4, jest to funkcja od prędkości w Machach. Natomiast w celu uzyskania bardziej dokładnych wartości została ona dalej interpolowana z użyciem skryptu `cd(machnum)_interpolation.py`.



Rysunek 1: Po interpolacji funkcji

Drugą wartoscią którą jest relatywnie ciężko policzyć jest gęstość powietrza, ponieważ zmienia się ona znacznie z wysokością. Tutaj korzystam z wzoru zaczerpniętego z[1]

$$\rho = \frac{pM}{RT} = \frac{pM}{R(T_0 - Lh)} = \frac{p_0 M}{RT_0}\left(1 - \frac{Lh}{T_0}\right)^{\frac{gM}{RL}-1} \tag{4}$$

Gdzie:

M - masa molowa (0.0289652 kg/mol dla powietrza)

R - stała gazowa (8.31446 J/(mol·K))

T - temperatura (w Kelwinach)

p - ciśnienie (w Pascalach)

L - zmienność temperatury z wysokością (0.0065 K/m)

g - przyspieszenie ziemskie

h - wysokość nad platformą startową

Zmienne z indeksem 0 oznaczają wartości w miejscu platformy startowej. W programie używam najbardziej rozbudowanego wzoru, jednak z naszą aparaturą pomiarową można zamienić na ten prostszy, z gotowym ciśnieniem które mamy dostępne.

Pozostaje jeszcze wyliczenie prędkości dźwięku c dla danej wysokości (gdyż uzależnione jest to od temperatury powietrza), tym razem zaczerpnęte z[2]

$$c = \sqrt{\gamma \cdot R_* \cdot T} \tag{5}$$

Gdzie $\gamma = 1.4$ oraz $R_* = 287.058$ to pewne stałe, natomiast T to temperatura zmieniająca się z wysokością:

$$T = T_0 - Lh \tag{6}$$

Dzięki wyliczeniu prędkości w machach $v_{mach} = v/c$ możemy precyzyjnie określić C_d w zależności od wysokości i prędkości rakiety.

Pozostaje jeszcze obliczenie masy rakiety w chwili zamknięcia zaworu. Tutaj problemy są na tyle duże że przybliże to zwykłą funkcją liniową, gdzie przy starcie masa całkowita jest sumą masy rakiety i paliwa, natomiast po czasie wypalenia silnika masą samej rakiety. Czas wypalenia zaczerpnę z pliku .eng z testu silnika.

[1] *Density of air*. Sierp. 2021. URL: https://en.wikipedia.org/wiki/Density_of_air#Variation_with_altitude.

[2] *Mach number*. Paź. 2021. URL: https://en.wikipedia.org/wiki/Mach_number#Calculation.

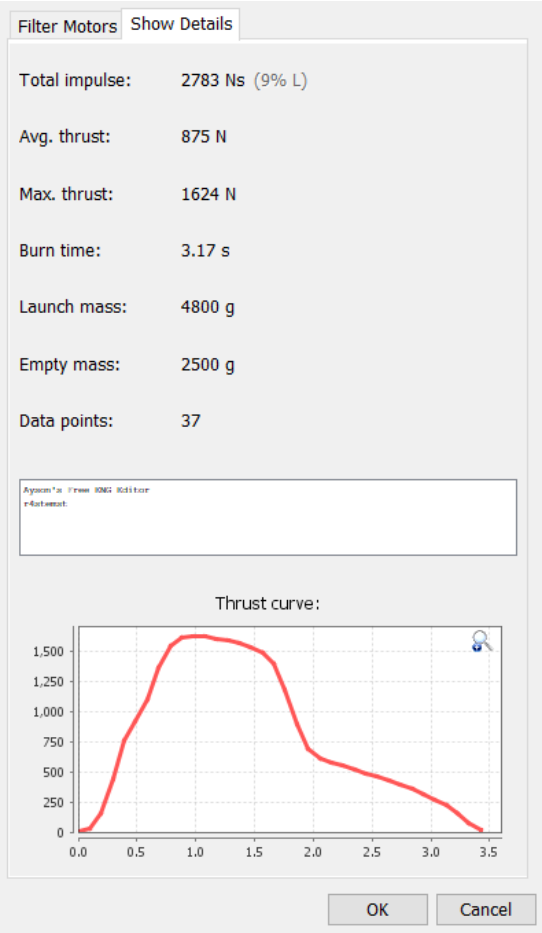
3 Testy

Do testów używam programu napisanego w C++, z danymi z rakiety R4S, LRE które przetworzyłem w taki sposób żeby wartości były w odstępach co 0,1s. Kod do interpolacji również daje do dyspozycji.

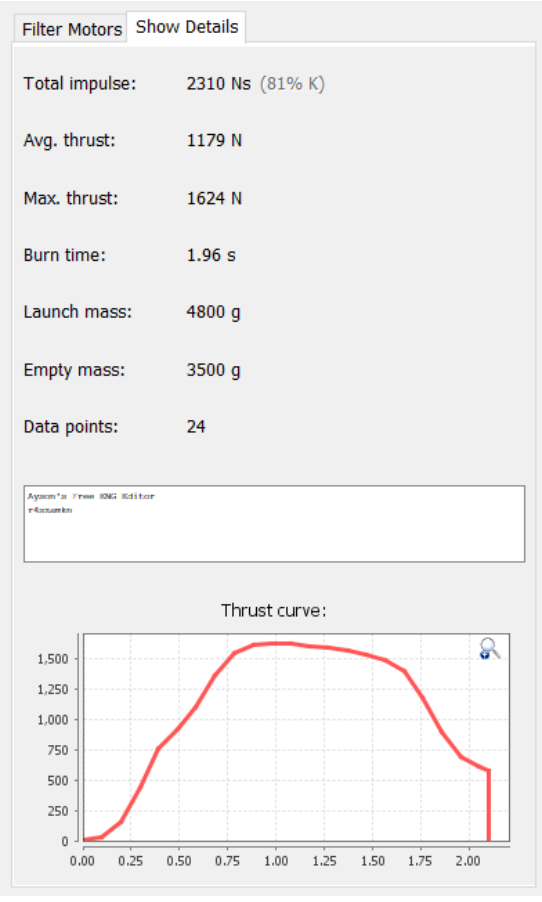
Zamysł jest prosty: ustawiam program tak, żeby wyznaczył mi czas urwania ciągu żeby rakieta doleciała na wysokość 800 metrów, zamiast realnych 1200. Następnie ucinam ciąg w pliku .eng w tej chwili (co ma odzwierciedlać zamknięcie zaworu) i sprawdzam jaką wysokość da mi OpenRocket dla takiego uciętego ciągu.

Wyniki są zadowalające, osiągnięte apogeum w OR było blisko, jednak trzeba tutaj wymienić parę rzeczy:

1. Ciąg był ucinany 'skokowo' co mało ma się do rzeczywistych warunków.
2. Trzeba wziąć pod uwagę że zawór zamyka się ok. 0.5s i ciąg nie ustaje od razu
3. Jest to narazie tylko jedna symulacja, trzeba przeprowadzić ich parę



Rysunek 2: Ciąg nieucięty



Rysunek 3: Ciąg ucięty

4 Kod programu

W razie jakby kod gdzieś się zgubił lub nie było do niego dostępu jest wrzucony tutaj

```
#include <iostream>
#include <fstream>
#include <cmath>
#define P0 101325             // sea level standard pressure,  can be changed to pressure on the launchpad, Pascal
#define T0 288.15             // sea level standard temperature, can be changed to temperature on the launchpad, Kelvin
#define L 0.0065              // temperature lapse rate, kelvin/meter
#define G 9.80665             // gravitational acceleration, meter/second^2
#define R 8.31446             // ideal gas constant, Jule/(mol*Kelvin)
#define M 0.0289652           // molar mass of dry air, kilogram/mol
#define GM_OVER_RL 5.25593278 // GM/RL, constant and used in lots of derivations
#define GAMMA 1.4             // ratio of specific heat of a gas at a constant pressure to heat at a constant volume for air
#define RSTAR 287.058         // specific gas constant of air, = R/M, https://en.wikipedia.org/wiki/Gas_constant#Specific_gas_constant
#define AREF 0.028            // reference area of rocket
#define TIMESTEP 0.1
#define TIMESTEPSQ 0.01

float CdOverMach[101]; // array for Cd values,
// CdOverMach[1] contains Cd value for Mach 0.01, CdOverMach[2] for Mach 0.02 and so on

float calculateTemperature(float height)
{
  return T0 - L * height;
}

/*! P = P0 * (1 - L*height/T0) ^ (GM/RL)
*  https://en.wikipedia.org/wiki/Density_of_air#Variation_with_altitude\
*  !!!! Pressure can also be taken from our measurements !!!!
*/
float calculatePressure(float height)
{
/* read pressure from main comp,
  return pressure
*/
  return P0 * pow((1 - L * height / T0), GM_OVER_RL);
}

/*!
*  https://en.wikipedia.org/wiki/Density_of_air#Variation_with_altitude
*/
float calculateAirDensity(float height)
{
  return calculatePressure(height) * M / (R * calculateTemperature(height));
}

/*!
*  https://en.wikipedia.org/wiki/Mach_number#Calculation
*/
float calculateSpeedOfSound(float height)
```

```
{
  return pow(GAMMA * RSTAR * calculateTemperature(height), 0.5);
}

/*!
*  https://en.wikipedia.org/wiki/Mach_number#Calculation
*/
float calculateMachNumber(float height, float velocity)
{
  return velocity / calculateSpeedOfSound(height);
}

float getCd(float machNumber)
{
  float machTimes100 = machNumber * 100.0;
  return CdOverMach[(int)machTimes100]; //conversion from float to int*100
}

float calculateDragForce(float height, float velocity)
{
  return calculateAirDensity(height) * pow(velocity, 2) * AREF * getCd(
    calculateMachNumber(height, velocity)) * 0.5;
}

int main()
{
CdOverMach[0] = 0.0; // for 0 Mach velocity
int i;
std::ifstream data;
data.open("data.txt");
for (i = 1; i < 101; ++i) // save values from data sheet
{
  data >> CdOverMach[i];
}
data.close();

std::ifstream flight;
flight.open("lot.txt");
float flightData[2][400];

for (i = 0; i < 400; ++i)
{
  flight >> flightData[0][i];
  flight >> flightData[1][i];
  std::cout << flightData[0][i] << " " << flightData[1][i] << std::endl;
}

// MAIN PROGRAM STARTS HERE //
bool running;   // for the while loop
/*
* velocity, height, dragForce - self explanatory
* simTime - time when data was taken and simulation began
* thrustEndTime - duration of engine working
* simHeight - height in timestamp n-1 and n respectively
* rocketMass - mass with motors, without propelant
* propellantMass - mass of propellant at the time of launch
* allMass - rocketMass + mass of propellant in time t
```

```
105 */
106 float velocity, height, dragForce, simTime, thrustEndTime = 3.423, simHeight
        [2], rocketMass = 13.141, propellantMass = 2.500, allMass; // mass in kg
107 for (i = 1; i < 400; ++i)
108 {
109   height = flightData[1][i];
110   velocity = (flightData[1][i] - flightData[1][i - 1]) / 0.1; // v = (h1 - h0)/
        dt
111
112   running = 1;
113
114   simTime = flightData[0][i];
115   simHeight[1] = flightData[1][i];
116   simHeight[0] = flightData[1][i - 1];
117   if (simTime < thrustEndTime)
118   allMass = rocketMass + propellantMass * ((thrustEndTime - simTime) /
        thrustEndTime);
119   else allMass = rocketMass;
120   while (running)
121   {
122     velocity = (simHeight[1] - simHeight[0]) / TIMESTEP;
123     if (velocity < 0)
124       {
125       running = 0;
126       if (simHeight[1] > 800.0)
127       std::cout << simHeight[1] << " <- height | time of turnoff -> " <<
        flightData[0][i] << " allMass = " << allMass << std::endl;
128       }
129     /*!
130     it would be good to include sth like this:
131
132     if (openedPilotParachute || reachedApogee)
133       running = 0;
134     */
135     dragForce = calculateDragForce(simHeight[1], velocity);
136     simHeight[0] = simHeight[1];    // height in t(n) prepare for next step
137     simHeight[1] = simHeight[1] + velocity * TIMESTEP - 4.9 * TIMESTEPSQ -
        dragForce / allMass * TIMESTEPSQ * 0.5; // height in t(n+1)
138     simTime += TIMESTEP;   // increase simTime
139     }
140
141     }
142 }
```

Listing 1: "Kod programu w C++"

Bibliografia

[1] *Density of air*. Sierp. 2021. URL: https://en.wikipedia.org/wiki/Density_of_air#Variation_with_altitude.

[2] *Mach number*. Paź. 2021. URL: https://en.wikipedia.org/wiki/Mach_number#Calculation.