Mathematical analysis of Pilot Trainer software

Adam Elkins, 29 January 2016

# Introduction

The pilot trainer software is required to receive a signal, interpret it as a throttle setting, then compute the resultant thrust from the engine, change in velocity and change in altitude assuming certain constants for engine performance.

Newton’s second law applied to a rocket can be written as

where in the first equality we compute the thrust acting on the engine by expelling propellant and in the second equality we obtain the response (acceleration) of the rocket itself due to engine thrust. By taking *vexh* = *Ispg0*, cancelling d*t* and rearranging, we obtain

which integrates analytically to give the rocket equation, ∆*V* = *Ispg0*ln(*M0*/*Mb*). The software must integrate this numerically, then use the δ*v* obtained to update the vehicle velocity and altitude.

# Calculations

A prototype program, simulator.c, was written in ANSI C (very similar to Arduino C) to demonstrate implementation of the necessary equations. First an appropriate time increment *dt* is set, then the propellant flow rate demanded by the throttle (in kg/s) is obtained and multiplied by *dt* to obtain *dm*, the fuel consumed in that time increment. This fuel increment is then subtracted from the vehicle mass to simulate propellant being consumed. The δ*v* due to thrust is then computed by calculating *thrust* = *Ispg0dm*/*m*.

To obtain the full *dv* an if loop is used to determine if the vehicle is on the ground or not and if the engine thrust is enough for lift-off; if so, a ground reaction is calculated as the difference between engine thrust and gravity, otherwise it is set to 0. The value of *dv* is then calculated by adding the thrust and ground reaction and subtracting gravity. This value is then added to the vehicle velocity. The change in vehicle altitude is then calculated from the vehicle velocity, and the time incremented. All this is wrapped in a while loop, which runs from *t* = 0 until a maximum time is reached. The program prints the altitude, velocity and time to *stdout* in each loop, which could be used to plot graphs of the vehicle flight profile.

# Code performance

The code appears to print sensible values to stdout. At the current fuel rate setting of 0.17 kg/s, simulating a constant throttle setting, the vehicle initially remains stationary before lifting off at 3.54 seconds and climbing slowly to 4.6 metres at 1.6 m/s after 12 seconds.

As implemented currently, the code occasionally returns negative values for propellant mass and altitude (when these are printed to stdout), which are unphysical. Adjustments to the code to look for this and return the appropriate result would eliminate this problem.

The code currently consumes very little propellant to fly, although this might be because it is simulating a relatively undemanding flight profile. This will need to be investigated to ensure that the code better reflects actual vehicle performance.

The code performs a simple rectangular numerical integration; this should be reasonably accurate for a sufficiently small value of *dt* but a trapezium integration would be better.

The calculation of altitude is suspect and needs to be looked at again.

# Conclusions

The code successfully demonstrates the implementation of the maths needed to simulate the vehicle flight; being written in ANSI C it will be readily converted to Arduino code when the time comes to implement it on the simulator itself. Several minor issues in this first version of the code have been identified for remediation.

# Appendix A: simulator.c

/\* Skeleton code for Lunar Hopper GDP Pilot Training Simulator

Adam Elkins, 28/01/2016

\*/

#include<stdio.h>

float fuelrate(void) {

/\* return fuel consumption rate (i.e. throttle setting) \*/

float rate = 0.17; /\* kg/s, simulate constant throttle setting \*/

return rate;

}

int main(void) {

/\* Declare variables \*/

int maxtime = 12; /\* length of run \*/

float elapsed\_time = 0; /\*\*/

float dt = 0.02; /\* time increment in seconds \*/

int Isp = 220; /\* specific impulse \*/

float m\_fuel = 4 + 6; /\* oxidiser plus fuel grain \*/

int m\_structure = 38 - m\_fuel;

float dm = 0; /\* propellant increment \*/

float g = 9.81; /\* gravity \*/

float s = 0; /\* displacement, spacecraft altitude \*/

float v = 0; /\* spacecraft velocity\*/

float dv = 0; /\* velocity imcrement \*/

float thrust = 0; /\* thrust \*/

float R = 0; /\* ground reaction force \*/

int Reaction = 0; /\* reaction force required? \*/

while (elapsed\_time < maxtime) {

/\* determine vehicle state: \*/

if (s == 0 && v == 0) { /\* vehicle on ground \*/

Reaction = 1; /\* compute reaction force \*/

}

/\* else-if statements? \*/

if (s <= 0 ) { /\* vehicle landed \*/

s = 0; /\* s < 0 is unphysical \*/

/\* print landing velocity \*/

v = 0; /\* vehicle has landed \*/

Reaction = 1; /\* compute reaction force \*/

}

if (s > 0) { /\* vehicle in flight \*/

Reaction = 0; /\* vehicle flying, no ground reaction \*/

}

/\* Compute sum of forces and accelerations: \*/

dm = fuelrate()\*dt; /\* simulate constant throttle setting \*/

if (m\_fuel > 0) {

m\_fuel -= dm; /\* reduce vehicle mass as fuel burnt \*/

}

else {

if (m\_fuel <= 0) { /\* ran out of fuel \*/

m\_fuel = 0;

dm = 0;

}

}

thrust = (Isp\*g\*dm)/(m\_fuel + m\_structure);

if (s <= 0) { /\* vehicle on ground \*/

if (thrust < g\*dt) { /\* vehicle not lifting off \*/

R = g\*dt - thrust; /\* calculate ground reaction force \*/

}

}

else {

R = 0;

}

dv = thrust - g\*dt + R;

v += dv;

s += v\*dt; /\* I need to double-check this \*/

elapsed\_time += dt;

printf("altitude %f velocity %f time %f \n", s, v, elapsed\_time);

}

return 0;

}