Pilot Trainer Progress Report

Adam Elkins, 8 February 2016

# Introduction

The ANSI C code written to implement the mathematics of the pilot training simulator was required to be adapted to run on an Arduino as a prerequisite to the completion of the trainer itself. The code was then required to obtain a throttle setting and use that to simulate controlling the engine thrust. This simply required wiring a potentiometer to one of the analogue inputs then adapting the existing code to read the input signal and convert it into a fuel flow rate.

Based on input from Dr Roberts, a second version of the ANSI C code was written that computed actual forces acting on the hopper, then computed the acceleration, velocity and displacement. A Python script was written to plot the results and the output of both versions was compared.

Copies of the Arduino code, improved ANSI C code, ant the python script are included in appendices A, B and C respectively.

# Arduino code

To implement the next step of development of the trainer, an Arduino was configured with a potentiometer connected to one of the analog inputs. The existing ANSI code was copied into a new file and adapted to run on the Arduino; this involved replacing the printf commands with Serial.print or Serial.println. This code was contained within a function called simulator(), which was called from loop().

A separate function throttle() was written to obtain the reading of the analog input and convert it into a fuel rate, called from the code in simulator(). This performed a calculation to obtain the theoretical maximum fuel rate by dividing the maximum thrust by the exhaust velocity (calculated by computing *Ispg0*) then multiplied this by the input reading. Print statements were written in loop() to print the velocity and acceleration to the serial monitor so that the vehicle flight could be seen, and code was added to determine the time of each loop and compute an appropriate delay so that the program ran in real time.

Once the code was finalised and running on the Arduino, testing verified that the code was able to read the signal from the potentiometer and interpret it as a throttle setting; thus the flight of the simulated hopper could be controlled. Initially the code was set to run for a long duration 100 seconds; in this time multiple simulated take-offs and landings could be achieved. It was found that the best way to fly the simulator was to keep the velocity low, to ±1 m/s, so that vehicle altitude and landing velocity could be controlled. It is noted that the hopper itself does not currently support a velocity reading, however.

Since the hopper’s actual flight will only last 10-12 seconds the code will need to be modified to end the flight after this time.

# Improved code mathematics

Following feedback, the original code was re-written to compute the forces acting on the vehicle rather than the delta-v that would result (which would be unphysical in the case of the ground reaction force). This involved computing the thrust and weight using standard formulas, computing the ground reaction force (if any), then computing the acceleration by summing the forces and dividing by vehicle mass. Values of altitude s and velocity v were then calculated from the normal equations of motion; the altitude being computed first since that relied on the value of the velocity before the increment of acceleration was applied.

Both versions of the code were then modified to separate the output into two streams, one of which contained pure numerical data that could be sent to a text file, with the other receiving the original formatted data.

# Python script and comparison of implementations

A short python script has been developed to plot the flight of the vehicle as computed by the simulator code. This was achieved by passing the output of the code to the command line into a text file, then reading the text file and plotting the data using standard Pylab functions. The output from both versions of the code was recorded and plotted; two graphs that resulted were identical showing that the two versions of the code produced identical results. These graphs are reproduced below.

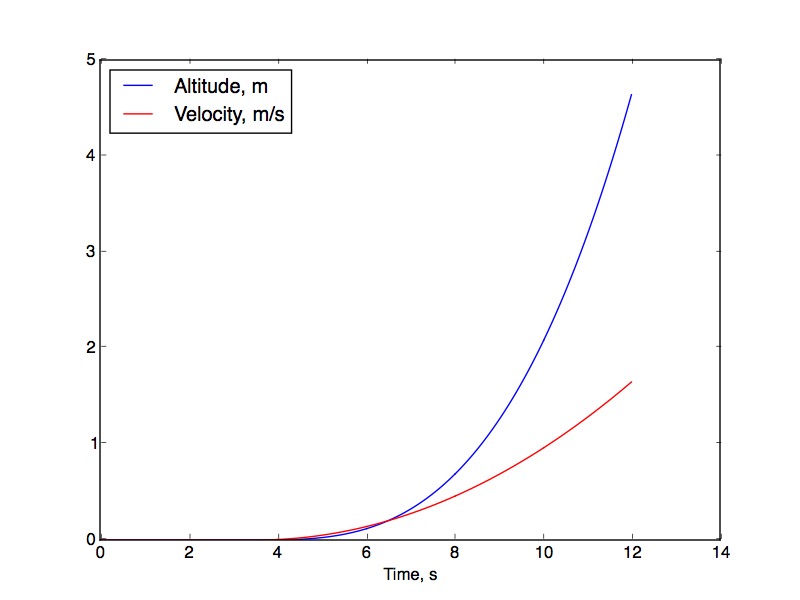


Figure : plot of vehicle velocity and altitude, original code

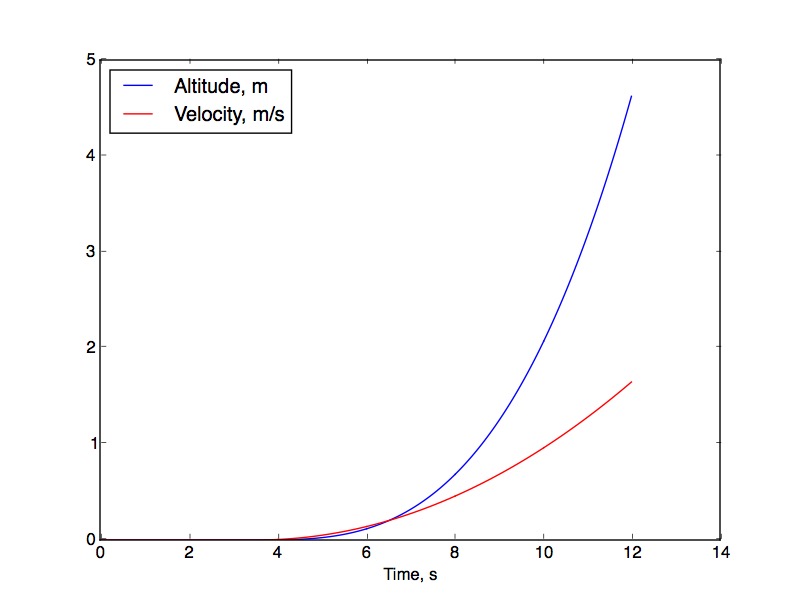


Figure : plot of vehicle velocity and altitude, new code

# Conclusions

The existing Arduino implementation already proves the acquisition of a throttle signal and its use to control the hoper simulator. The code modified to compute forces acting on the vehicle is shown to produce identical results to the earlier code; this code will be used going forward as it is a more accurate physical representation of the system.

# Appendix A: Arduino code

/\* simulator.ino

\* Code for Lunar hopper Pilot Training Simulator: obtain a

\* throttle setting reading from a potentiometer connected to

\* analog input and compute resulting vehicle flight velocity and

\* altitude.

\*

\* Adam Elkins, 03/02/2016

\*/

// Declare variables and constants here

const int throttlePin = 2; // Declare throttle pin

boolean prnt = true; // true to print to serial monitor

const float dt = 0.02; // time step in milliseconds

const int maxtime = 100; // length of run

float elapsed\_time = 0; //

const int Isp = 160; // specific impulse

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

const int m\_structure = 28; // mass of structure

float dm = 0; // propellant increment

const float g = 9.81; // gravity

float s = 0; // displacement, spacecraft altitude

float v = 0; // spacecraft velocity

float dv = 0; // velocity imcrement

float dv\_thrust = 0; // dv due to thrust

float R = 0; // ground reaction force

int startTime = 0;

int endTime = 0;

int delayTime = 0; // variables to compute correct delay time

float maxfuelrate;

float fuelrate;

float throttle() {

maxfuelrate = 400/(Isp\*g); // = f/v\_exh

fuelrate = maxfuelrate\*analogRead(throttlePin)/1023;

return fuelrate;

}

void setup() {

//

if (prnt) {

Serial.begin(115200);

}

}

void loop() {

//

startTime = millis();

if (prnt) {

Serial.print("altitude ");

Serial.print(s,9);

Serial.print(" velocity ");

Serial.print(v,6);

Serial.print(" fuel ");

Serial.print(m\_fuel);

Serial.print(" time ");

Serial.print(elapsed\_time);

Serial.print(" millis ");

Serial.print(millis());

Serial.print("\n");

}

simulator();

endTime = millis();

/\* to compute dynamic delay:

\* get millis() at start of loop

\* get millis at end of loop and subtract to get difference

\* delayTime = dt - (difference)

\*/

delayTime = int(dt\*1000) - (endTime - startTime);

//Serial.println(delayTime);

delay(delayTime);

}

void simulator(){

if (elapsed\_time < maxtime) {

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

dm = throttle()\*dt; /\* obtain throttle setting \*/

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

dm = m\_fuel;

m\_fuel = 0;

}

else {

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

}

/\* Compute dv due to thrust: \*/

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

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

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

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

}

}

else { /\* Vehicle in flight \*/

R = 0; /\* Zero reaction force \*/

}

dv = dv\_thrust - g\*dt + R; /\* Calculate velocity increment \*/

v += dv; /\* increment vehicle velocity \*/

if (s + v\*dt < 0) { /\* Landing or on ground \*/

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

if (v != 0) {

Serial.print("Landing; impact speed = ");

Serial.println(v); /\* print landing speed \*/

}

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

}

else { /\* Vehicle in flight \*/

s += v\*dt; /\* Increment vehicle altitude \*/

}

elapsed\_time += dt; /\* Increment time \*/

/\* Print vehicle altitude, velocity and time: \*/

}

}

# Appendix B: Improved ANSI C code

/\* simulator3a.c

Code for Lunar Hopper GDP Pilot Training Simulator, improved to

compute actual forces acting on the hopper and the resulting acceleration

Adam Elkins, 08/02/2016

\*/

#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 Thrust = 0;

float Weight = 0;

float Reaction = 0;

float a = 0;

while (elapsed\_time < maxtime) {

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

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

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

dm = m\_fuel;

m\_fuel = 0;

}

else {

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

}

/\* Compute forces acting on vehicle, obtain resultant and acceleration \*/

Thrust = Isp\*g\*dm/dt; /\* Exhaust velocity (Isp\*g)\*mdot \*/

Weight = (m\_fuel + m\_structure)\*g;

if (s <= 0) { /\* Vehicle is on ground, compute ground reaction force \*/

if (Thrust < Weight) {

Reaction = Weight - Thrust;

}

}

else {

Reaction = 0;

}

/\* compute acceleration: sum of forces divided by mass \*/

a = (Thrust + Reaction - Weight)/(m\_fuel + m\_structure);

/\* Compute altitude s = s0 + u\*t + 0.5\*a\*t^2: \*/

s += v\*dt + 0.5\*a\*dt\*dt;

/\* Compute velocity v = u + a\*t: \*/

v += a\*dt;

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

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

if (v != 0) {

fprintf(stderr, "Landing; impact speed = %f\n", v); /\* print landing speed \*/

}

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

}

elapsed\_time += dt; /\* Increment time \*/

fprintf(stdout, " %2.9f\t%f\t%f\n", s, v, elapsed\_time);

fprintf(stderr, "altitude %2.9f velocity %f acceleration %f time %f \n", s, v, a, elapsed\_time);

}

/\* Print final altitude and velocity \*/

fprintf(stderr, "Final altitude %2.9f \nFinal velocity %f \n", s, v);

return 0;

}

# Appendix C: Python graph plotting script

"""plotsimulator.py

Python script to obtain numerical data saved from simulator code

and plot a graph of velocity and altitude against time

Adam Elkins, 08/02/2016"""

import pylab as p

filein = open("simulatordata.txt", 'r')

lines = filein.readlines()

filein.close()

rows = [line.split() for line in lines]

Altitude = [row[0] for row in rows]

Velocity = [row[1] for row in rows]

Time = [row[2] for row in rows]

p.plot(Time, Altitude, 'b', label='Altitude, m')

p.plot(Time, Velocity, 'r', label='Velocity, m/s')

p.legend(loc='upper left')

p.xlabel('Time, s')

p.savefig('Simulatorplot.jpg')

p.show()