Landing a Runabout

In this lesson we will discuss the issue of constructing a model for the landing of a runabout from a parent spacecraft onto the surface of a planet. We will develop the modeling equations for controlled landing assuming a force of gravity on the given planet. We will also derive an algorithm for the piloting of the runabout.

The objective of this lesson is to provide a framework for the introduction of conditional loops with C++.

1.1 The Physical Situation

The laws of gravity are sufficient to describe the motion of a craft as it travels towards the surface of the planet. Without any other force that counteracts gravity, any object under the influence of gravitational forces will be in free fall.

The pilot can land the runabout safely by directing its thrusters to produce a force (thrust) that counteracts the gravitational pull exerted by the planet. The net acceleration of the runabout is directly related to the amount of fuel consumed.

The thrusters consume fuel in proportion to the thrust exerted to counteract gravity. Because it is a runabout designed for limited travel, the amount of fuel is limited. Further, the runabout thrust capacity is also limited by its design. The pilot’s objective is to land the runabout softly on the surface of the planet. Softly means with a small enough velocity to be absorbed by the runabout structure and landing mechanism.
1. Landing a Runabout

1.2 The Model

The laws of gravity describe the motion of an object as it falls towards the surface of a planet. The pilot counteracts the acceleration of the planet by directing the consumption of fuel by the runabout thrusters. The pilot’s objective is to land the runabout softly on the surface of the planet.

The runabout thrusters consume fuel to produce a force that counteracts the gravitational pull exerted by the planet. The net acceleration of the runabout is directly related to the amount of fuel consumed. Let us assume that \( g \) is the gravity of the planet, and \( f \) is the amount of fuel-per-unit-time consumed by the runabout. Then, the net acceleration \( a \) satisfies the following relationship.

\[
a = g + 0.35 \times f
\]

Consider the time \( t = 0 \) as the reference time when the runabout makes its initial approach to the planet. At this reference time the altitude of the runabout over the planet surface is \( h(0) \) and its (vertical) velocity is \( v(0) \).

The relation over a time span \([0, t]\) of the velocity \( v(t) \) and the net acceleration \( a \) is obtained using simple integration in time.

\[
v(t) = v(0) + \int_0^t a \, \zeta. \]

Similarly the altitude is given by:

\[
h(t) = h(0) + v(0) \times t + \int_0^t \int_0^\xi a \, \zeta \, d\zeta. \quad (1.1)
\]

1.2.1 Model per Time-Unit

The pilot provides fuel per unit-time. In the acceleration equation above, the value of \( f \) changes as the pilot decides to use more or less fuel. However, \( f \) is constant per each time unit. Hence \( a \) is constant per each time unit. Therefore, the equations need to be rewritten on a per unit-time. The integrations in equations for velocity and altitude should be completed over \([t, t + 1]\). Let \( 0 < \tau \leq 1 \), then

\[
v(t + \tau) = v(0) + \int_0^{t+\tau} a \, \zeta
\]

\[
= v(0) + \int_0^t a \, \zeta + \int_t^{t+\tau} a \, \zeta
\]

\[
= v(t) + a \times (t + \tau - t)
\]

\[
= v(t) + a \times \tau. \quad (1.2)
\]

Similarly,

\[
h(t + \tau) = h(0) + v(0) \times (t + \tau) + \int_0^{t+\tau} \int_0^\xi a \, \zeta \, d\zeta
\]

\[
= h(0) + v(0) \times t + v(0) \times \tau + \int_0^t \int_0^\xi a \, \zeta \, d\zeta + \int_t^{t+\tau} \int_0^\xi a \, \zeta \, d\zeta
\]

\[
= h(t) + v(0) \times \tau + \int_t^{t+\tau} (\int_0^\xi a \, \zeta + \int_0^\xi a \, \zeta) \, d\zeta
\]
\[ h(t) + v(0) \times \tau + \int_{t}^{t+\tau} (v(t) - v(0))d\xi + \int_{t}^{t+\tau} a \times (\xi - t)d\xi \]
\[ = h(t) + v(0) \times \tau + (v(t) - v(0)) \times \tau + a \times \frac{(t + \tau)^2}{2} - t(t + \tau) - \frac{t^2}{2} + t^2 \]
\[ = h(t) + v(t) \times \tau + \frac{a \tau^2}{2}. \quad (1.3) \]

### 1.2.2 Landing

The pilot is in control as long as the altitude \( h(t) \) remains greater than zero. In order to soften the landing the pilot acts to expend some fuel. The landing occurs when the runabout touches the surface of the planet, \( h(t) = 0 \). The landing can occur sometime in a time-unit time span \([t, t+1]\). Let \( t + \delta t \) be actual time of touch down for some \( \delta t > 0 \). Hence \( \delta t \) must satisfy:

\[ 0 = h(t + \delta t) = h(t) + v(t) \times \delta t + \frac{a \delta t^2}{2}. \]

A quadratic equation for \( \delta t \). Therefore,

\[ \delta t = \frac{-v(t) \pm \sqrt{v(t)^2 - 2a \times h(t)}}{a}. \]

Note that the quadratic equation has two possible solutions. Using the modified method for the quadratic equation,

\[ \delta t_1 = \frac{-v(t) - \text{sign}(v) \sqrt{v(t)^2 - 2a \times h(t)}}{a}. \]

Note that the speed of descend \( v(t) \) is a negative quantity since the axis is positive upwards and the runabout is descending. Hence,

\[ \delta t_1 = \frac{-v(t) + \sqrt{v(t)^2 - 2a \times h(t)}}{a}, \]

and

\[ \delta t_2 = \frac{2h(t)}{a \delta t_1}. \]

The desired \( \delta t \) is the smallest positive root of \( 0 = h(t + \delta t) \). When \( a < 0 \), \( \delta t_1 < 0 \) which is not physical. When \( a > 0 \), \( 0 < \delta t_2 < \delta t_1 \). Thus, \( \delta t_2 \) is the desired root in both cases. Hence:

\[ \delta t_1 = \frac{-v(t) + \sqrt{v(t)^2 - 2a \times h(t)}}{a}, \]

and

\[ \delta t = \frac{2h(t)}{a \delta t_1}. \]

Once the landing time is determined, the landing velocity can also be determined.

\[ v(t + \delta t) = v(t) + a \times \delta t. \]

If the landing velocity is sufficiently small for the structure of the runabout to sustain the impact, the pilot has succeeded in a soft landing. Otherwise, the runabout will be heavily damaged and hands may be lost.
1.3 Method

Each time unit the pilot must repeat the decision of how much fuel to consume. The following algorithm gives the decision making process for landing.

**Algorithm: Runabout Landing**

*Input:*

- \( h(0) \), the unital altitude in m.
- \( v(0) \), the initial velocity in m/s.
- \( Cf \), the fuel capacity in fuel units.

*Initialize*

- \( Rf = Cf \), Amount of fuel remaining.
- \( t = 0 \).

*The Landing*

- Repeat while \( h(t) > 0.0 \)
  
  - Determine \( f(t) \), the amount of fuel per unit-time to be consumed.
  
  - \( Rf = Rf - f(t) \).
  
  - \( v(t + 1) = v(t) + a \).
  
  - \( h(t + 1) = h(t) + v(t) + a/2 \).
  
  - if \( h(t + 1) < 0.0 \) then
    
    \[
    \delta t = \frac{-v(t) + \sqrt{v(t)^2 - 2a \times h(t)}}{a},
    \]
    
    \[
    \delta t = \frac{2 \times h(t)}{a \times \delta t}.
    \]
    
    - \( v(t + \delta t) = v(t) + a \times \delta t \).
1. Landing a Runabout

- else
  
  * \( t = t + 1 \)

- end repeat

- if \( (v(t) > -5.5 \text{ m/s}) \) then

  - Landed Softly!!

- else

  - Crashed on landing! All hands lost.

- endif

This algorithm requires that some initial data be provided. Further, during flight, it requires that the amount of fuel used per unit-time be also provided. These are provided by the pilot at the appropriate time.

1.4 Implementation

The implementation of the simulation of the Landing Algorithm requires several constructs in the language that is chosen to do the implementation. The design consists of constructing functions that request the user to provide interactively the initial data, and the fuel consumption. These are to be written as functions that return the appropriate values. the Landing Algorithm is also to be implemented as a function that produces a lot of output as it goes through the flight. It will return the impact velocity so the calling program will be able to make the decision on whether it landed safety or not.

The language used for implementation requires several control statements to implement the Repeat While and if-then-else constructs in the algorithm.

1.4.1 Implementation in C++

The implementation of the Landing Algorithm will be presented in the C++ programming language. C++ is increasing in popularity. This lecture notes attempt to illustrate some of these reasons.

This lesson provides a framework for the introduction of conditional loops with C++.

The main program

Here is the main() function for the pilot simulation program.
// BEGIN file runabout.cpp
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

#include "lander.h" // includes declarations of all necessary functions

int main() // This is the main function for the pilot program.
{
    /* First declare the variables that are needed */
    float planet_gravity, initial_altitude, initial_velocity; //initial values
    float initial_fuel, impact_speed; //initial values

    /* Initialize variables */
    planet_gravity = get_gravity();
    initial_altitude = get_altitude();
    initial_velocity = get_velocity();
    initial_fuel = get_fuel_capacity();

    /* Obtain landing speed */
    impact_speed = landing_speed( planet_gravity,
                                 initial_altitude, initial_velocity, initial_fuel);
    if( impact_speed >= -5.5 ) {
        cout << "Safe landing, Congratulations" << endl; // All is well.
    } else{
        cout << "Runabout has crashed. All hands lost." << endl; // Get a new pilot.
    }
    return 0;
}
// END file runabout.cpp

Note that the expressions get_gravity(), get_altitude(), get_velocity(), and get_fuel_capacity() are function calls to functions that require no input. These and the function landing_speed() are declared in the file lander.h.

C++ implementation of the Landing Algorithm

Now we turn our attention to the rest of the landing Algorithm. The main function calls a the function landing_speed with four arguments.

// BEGIN file lander.cpp
#include <cmath>
#include "lander.h" //

#include <iostream>
using std::cout;
using std::cin;
using std::endl;


float landing_speed(float planet_gravity, float initial_altitude, 
    float initial_velocity, float initial_fuel)
{
    // Implementation of the Landing Algorithm.
    
    float velocity; // Output variables
    float acceleration, altitude, deltat; // Local variables
    float velocitytp1, altitudetp1, remaining_fuel; // Local variables
    float fuel_to_be_consumed; // Local variables
    float time =0; //Local Variable
    
    altitude = initial_altitude;
    velocity = initial_velocity;
    remaining_fuel = initial_fuel;

    while ( altitude > 0.0 )
    {
        if (remaining_fuel > 0. )
        {
            fuel_to_be_consumed = fuel_consumption(remaining_fuel);
            remaining_fuel = remaining_fuel - fuel_to_be_consumed;
        } else
        {
            fuel_to_be_consumed = 0.;
        }
        acceleration = planet_gravity + .35 * fuel_to_be_consumed;
        velocitytp1 = velocity + acceleration ;
        altitudetp1 = altitude + velocity + acceleration/2.0;
        if ( altitudetp1 <= 0.0 ) {
            deltat=(-velocity+sqrt(velocity*velocity-2.0*acceleration*altitude)) /acceleration;
            deltat=(2.0*altitude) /(acceleration*deltat);  
            velocity = velocity + acceleration* deltat;
            altitude = 0.0;
            time = time + deltat;
        } else {
            time += 1;
            velocity = velocitytp1;
            altitude = altitudetp1;
        }
        cout << "Altitude = " << altitude<< ", Velocity = " <<velocity  
             "Acceleration = " << acceleration <<endl;
        cout << "Time = " << time "", Remaining fuel = " << remaining_fuel<< endl;
    }
    return velocity;
} // END file lander.cpp

There are two things to note in this function. The the while loop, and the += updating operation.
1. The while loop construct takes the form:

```cpp
while (condition ){
    ... // while-block of statements
    ...
}
```

The while-block of statements is executed every time through the loop as long as condition is True. The condition is checked before the execution of each loop. There are other forms of loop construct such as the do-while:

```cpp
do {
    ... // do-while-block of statements
    ...
} while (condition ) ;
```

The do-while-block of statements is executed every time through the loop as long as condition is True. The condition is checked after the execution of each loop.

2. C++ provides several updating operators: +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=. For each operator oper and a variable var,

```
a oper = expr;
```

is equivalent to

```
a = a oper expr;
```

For instance `a -= 7` is equivalent to `a = a - 7`.

The function `fuel_consumption()` is declared in the file `lander.h` and its definition is included in the file `pilot.cpp`.

**Fuel Consumption**

The code implementing the fuel consumption should check both the lower limits and the upper limits for the amount of fuel entered by the user. If it is not within the specified bounds, it is not acceptable and the user should be prompted to get a new value.

```cpp
// BEGIN file pilot.cpp
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

float fuel_consumption(float remaining_fuel) {
    float fuel;
    float limit;

    limit = 50.0;
```
if ( limit > remaining_fuel ) limit = remaining_fuel;
do {
    cout << "Enter fuel_consumption "
    cin >> fuel;
    if (fuel<0.0 || fuel>limit){
        cout << "Fuel_consumption must be between zero and"<<limit<< endl;
    }
} while (fuel < 0.0 || fuel>limit);
cout << "Fuel_consumption = " <<fuel <<endl;
return fuel;
}

Include here the definitions of the functions:
float get_velocity();
float get_altitude();
float get_gravity();
float get_fuel_capacity();
*
// END file pilot.cpp

The Other Functions

The declarations of the remaining functions are included in the file lander.h. All of the require input from the user.

// BEGIN file lander.h
float fuel_consumption(float remaining_fuel);
float get_velocity();
float get_altitude();
float get_gravity();
float get_fuel_capacity();

The definition of these functions is left as an as an exercise. These functions need to impose various conditions on the input.

- There are no restrictions on the numeric value of the velocity.
- The altitude must be greater than zero.
- The gravity must be a negative number.
- The fuel capacity must be greater than zero.

1.5 Assessment

Below are the results for two sets of values for landing the runabout.
1. Landing a Runabout

1.5.1 First Simulation

Enter Initial Altitude of Runabout Above Planet (m): 65
Enter Initial Vertical Velocity of Runabout (m/s): 0
Enter Gravitational Acceleration of Planet (m/s^2): -9.8
Enter Initial Fuel in Runabout (fuel units): 300

Initial Altitude = 65 m
Initial Velocity = 0 m/s
Planet’s Gravity = -9.8 m/s^2
Fuel Capacity = 300 fuel units

Time 0 s:
Altitude= 65 m, Velocity= 0 m/s, Fuel= 300 units.
   Enter fuel to be consumed this time period (fuel units): 0
   Fuel consumed this time period = 0 units

Time 1 s:
Altitude= 60.1 m, Velocity= -9.8 m/s, Acceleration= -9.8
Remaining Fuel= 300 units.
   Enter fuel to be consumed this time period (fuel units): 20
   Fuel consumed this time period = 20 units

Time 2 s:
Altitude= 48.9 m, Velocity= -12.6 m/s, Acceleration= -2.8
Remaining Fuel= 280 units.
   Enter fuel to be consumed this time period (fuel units): 50
   Fuel consumed this time period = 50 units

Time 3 s:
Altitude= 40.15 m, Velocity= -4.9 m/s, Acceleration= 7.7
Remaining Fuel= 230 units.
   Enter fuel to be consumed this time period (fuel units): 28
   Fuel consumed this time period = 28 units

Time 4 s:
Altitude= 35.25 m, Velocity= -4.9 m/s, Acceleration= 0
Remaining Fuel= 202 units.
   Enter fuel to be consumed this time period (fuel units): 28
   Fuel consumed this time period = 28 units

Time 5 s:
Altitude= 30.35 m, Velocity= -4.9 m/s, Acceleration= 0
Remaining Fuel= 174 units.
   Enter fuel to be consumed this time period (fuel units): 28
   Fuel consumed this time period = 28 units

Time 6 s:
Altitude= 25.45 m, Velocity= -4.9 m/s, Acceleration= 0
Remaining Fuel= 146 units.
   Enter fuel to be consumed this time period (fuel units): 28
   Fuel consumed this time period = 28 units
1. Landing a Runabout

Time 7 s:
Altitude= 20.55 m, Velocity= -4.9 m/s, Acceleration= 0
Remaining Fuel= 118 units.
Enter fuel to be consumed this time period (fuel units): 28
Fuel consumed this time period = 28 units

Time 8 s:
Altitude= 15.65 m, Velocity= -4.9 m/s, Acceleration= 0
Remaining Fuel= 90 units.
Enter fuel to be consumed this time period (fuel units): 28
Fuel consumed this time period = 28 units

Time 9 s:
Altitude= 10.75 m, Velocity= -4.9 m/s, Acceleration= 0
Remaining Fuel= 62 units.
Enter fuel to be consumed this time period (fuel units): 28
Fuel consumed this time period = 28 units

Time 10 s:
Altitude= 5.84999 m, Velocity= -4.9 m/s, Acceleration= 0
Remaining Fuel= 34 units.
Enter fuel to be consumed this time period (fuel units): 28
Fuel consumed this time period = 28 units

Time 11 s:
Altitude= 0.949986 m, Velocity= -4.9 m/s, Acceleration= 0
Remaining Fuel= 6 units.
Enter fuel to be consumed this time period (fuel units): 28
Fuel consumption must be between zero and 6
Reenter fuel consumption: 6
Fuel consumed this time period = 6 units

Time 11 s:
Deltat =4.54973e-09 s,
Altitude= 0 m, Velocity= -4.9 m/s, Acceleration= -7.7
Remaining Fuel= 0 units.
Impact at Time = 11 s
Impact Speed = -4.9 m/s

Safe landing, Congratulations

1.5.2 Second Simulation

This is a simulation of a landing runabout. You are the pilot.

Enter Initial Altitude of Runabout Above Planet (m): 200
Enter Initial Altitude of Runabout Above Planet (m): 200
Enter Initial Vertical Velocity of Runabout (m/s): -30
Enter Gravitational Acceleration of Planet (m/s^-2): -5
Enter Initial Fuel in Runabout (fuel units): 400
Initial Altitude = 200 m
Initial Velocity = -30 m/s
Planet's Gravity = -5 m/s^2
Fuel Capacity = 400 fuel units

Time 0 s:
Altitude= 200 m, Velocity= -30 m/s, Fuel= 400 units.

Enter fuel to be consumed this time period (fuel units): 0
Fuel consumed this time period = 0 units

Time 1 s:
Altitude= 167.5 m, Velocity= -35 m/s, Acceleration= -5
Remaining Fuel= 400 units.

Enter fuel to be consumed this time period (fuel units): 14.2857
Fuel consumed this time period = 14.2857 units

Time 2 s:
Altitude= 132.5 m, Velocity= -35 m/s, Acceleration= -5.24521e-06
Remaining Fuel= 385.714 units.

Enter fuel to be consumed this time period (fuel units): 14.2857
Fuel consumed this time period = 14.2857 units

Time 3 s:
Altitude= 97.5 m, Velocity= -35 m/s, Acceleration= -5.24521e-06
Remaining Fuel= 371.429 units.

Enter fuel to be consumed this time period (fuel units): 14.2857
Fuel consumed this time period = 14.2857 units

Time 4 s:
Altitude= 62.5 m, Velocity= -35 m/s, Acceleration= -5.24521e-06
Remaining Fuel= 357.143 units.

Enter fuel to be consumed this time period (fuel units): 50
Fuel consumed this time period = 50 units

Time 5 s:
Altitude= 33.75 m, Velocity= -22.5 m/s, Acceleration= 12.5
Remaining Fuel= 307.143 units.

Enter fuel to be consumed this time period (fuel units): 30
Fuel consumed this time period = 30 units

Time 6 s:
Altitude= 14 m, Velocity= -17 m/s, Acceleration= 5.5
Remaining Fuel= 277.143 units.

Enter fuel to be consumed this time period (fuel units): 50
Fuel consumed this time period = 50 units

Time 7 s:
Altitude= 3.24995 m, Velocity= -4.50001 m/s, Acceleration= 12.5
Remaining Fuel= 227.143 units.

Enter fuel to be consumed this time period (fuel units): 14.2857
Fuel consumed this time period = 14.2857 units

Time 7 s:
1. Landing a Runabout

\[ \Delta t = -2.77433 \times 10^{-6} \text{ s}, \]
\[ \text{Altitude} = 0 \text{ m}, \quad \text{Velocity} = -4.50001 \text{ m/s}, \quad \text{Acceleration} = -5.24521 \times 10^{-6} \]
\[ \text{Remaining Fuel} = 212.857 \text{ units}. \]
\[ \text{Impact at Time} = 7 \text{ s} \]
\[ \text{Impact Speed} = -4.50001 \text{ m/s} \]

Safe landing, Congratulations

1.5.3 Runabout Project

1. Copy the files `lander.cpp`, `pilot.cpp` and `runabout.cpp` under your `es2503Programs` directory.

2. In the file `pilot.cpp` provide definitions for the functions `get_gravity()`, `get_velocity()`, `get_altitude()` and `get_fuel_capacity()`. These functions shall ask the user to input the following values:
   - The initial altitude of the runabout above the planet, in m.
   - The initial vertical velocity of the craft, in m/s.
   - The gravity on this planet, in m/s\(^2\)
   - The initial fuel in the runabout, measured in fuel units.

   Your functions must enforce conditions on the initial values.
   - The initial altitude of the runabout must be greater than zero.
   - The gravity must be negative.
   - The initial fuel must be positive or zero.

   Your program must print an appropriate warning message when it encounters invalid input values. And it should repeat the query for input from the user.

3. Compile your program using
   \[ g++ \text{ runabout.cpp lander.cpp pilot.cpp} \ -o \text{ runabout} \]

4. When the acceleration is close to zero the quadratic equation is near a linear equation. Consider the case of the acceleration of gravity in the planet earth, -9.80. A thrust of 28 Fu (9.8/.35) will result in a net acceleration of 0.0 for a lander with a .35 conversion rate. When the acceleration is near zero the quadratic term in the equation is near zero and the equation becomes linear. In fact the quadratic formula may return inaccurate answers for small values of acceleration. In this case the following linear equation for \( \delta t \) should be used.

   \[ 0 = h(t + \delta t) = h(t) + v(t) \times \delta t \]

   When testing for small values of \( a \), test whether \( |a| + 1.0 \leq 1.0 \). If this is true \( a \) is very small and the equation could be considered linear. Modify your `lander.cpp` file to implement this safety device.

5. Once your program is working, test it out. In the Assessment section there are sample runs. Compare the behavior of your simulator to the behavior shown in the examples. If you input the indicated values, then your program should produce the same results.

   When you are confident that your simulator is working properly, figure out how to successfully land the runabout for each of the following sets of initial conditions:
   Condition Set 1:
1. Landing a Runabout

- Initial altitude: 200 m.
- Initial velocity: -30 m./s.
- Gravity: -5.0 m./s.²
- Initial fuel: 400 units

Condition Set 2:

- Initial altitude: 150 m.
- Initial velocity: 0 m./s.
- Gravity: -9.8 m./s.²
- Initial fuel: 1000 units

6. Modify your program so it prints out the input values and all the output is directed to the file `runaboutResults.txt`. Run it with Condition set 1.

7. Once the program works correctly, handin the files: `runabout.cpp`, `lander.cpp`, `pilot.cpp` and `runaboutResults.txt`.

8. Copy the file `runaboutResults.txt` to your public_html/es2503Reports directory.

9. Change directory to your public_html/es2503Reports directory, and create the report file `runaboutReport.html` which explains the project and includes relative link to the file `runaboutResults.txt`.

10. Modify your `es2503.html` to place links to `runaboutReport.html`.

11. Hand in using the `handin` command the following files: `runaboutReport.html` and `es2503.html`.

12. Clean up your home directory and any of your public areas to make sure that no files containing your program or portion thereof are accessible by any user.