Embedded electronics exam

J.-M Friedt, December 14, 2021

A shell is instrumented with a microcontroller to know its projection range after being ejected from a gun. The microcontroller is fitted with an accelerometer to deduce its position along its ballistic flight (no self-propulsion source) sampling one measurement every time step $dt = 1$ ms. Remember that the velocity v_n is the first integral of the acceleration a so that $v_{n+1} = v_n + a_n \cdot dt$ and that position is the first integral of the velocity so that $r_{n+1} = r_n + v_n \cdot dt$. Since these quantities are vectors, they each have two independent components in a cartesian coordinate systems (x, y).

The first set of questions are answered on the PC with the host computer compiler:

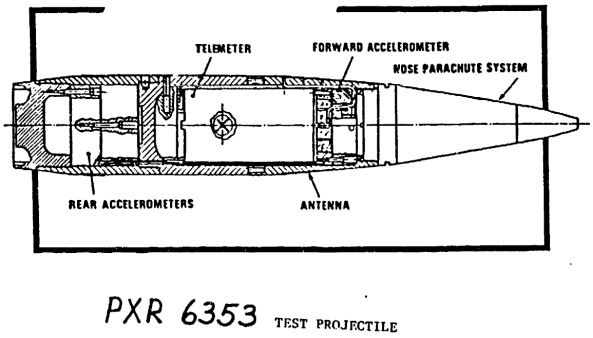

Figure 1: Measurement principle schematic, from [1]

1. How would you represent the shell state in C with a variable `shell` holding all current variables representative of its state at time t, using floating point number representations?
2. Considering the variable content `shell` has been initialized at time index n, provide a function in a file `shell_float.c` that returns the shell state at time $n + 1$ according to the equations provided in the introduction, considering that the acceleration is unchanged from one iteration to the other. Again remember that the two cartesian axis are processed independently of each other and following the same equation sequence.
3. Provide in a `main.c` function the declaration of the variable representing the shell and consider the following conditions: the acceleration is $a_n(x) = 0$ along the X-axis (no self propulsion), $a_n(y) = -10$ m.s^{-2} $\forall n$ along the Y direction due to gravity, the initial velocity is 1000 m/s at an angle of 45°, and the initial position of the shell is $(0, 0)$. Compile the program into an executable for the host PC.
4. If you did not do so previously, provide a Makefile automating the compilation of the executable, defining the state calculation filename `shell*` as a variable that will be changed easily in a single location when updated from floating point calculation to fixed point calculation (next section).
5. Iterate the calculation of the shell state until it falls back on the ground. What is this iteration ending condition? What are the values of the bullet state when this condition is reached, and most significantly at what range did it hit the ground?

The second set of questions are tested on the microcontroller emulator `qemu` found in `/home/jmfriedt/qemu_stm32/arm-softmmu`. Remember that a program `prog.bin` is emulated on Qemu with the following command line:

```
qemu-system-arm -M stm32-p103 -serial stdio -serial stdio -serial stdio -kernel prog.bin
```

Now that you reached the conclusion on the range reached by the shell, we wish for this calculation to be performed by the on-board microcontroller which happens to be an STM32

6. convert all floating point calculations into fixed point calculations with 3 accurate decimals in a new file `shell_fixed.c`. How many bits of the integer are dedicated to the fractional part when 3 accurate decimal digits are computed?
7. Modify your `Makefile` to compile the program using fixed point calculation so that it compiles and executes on the host PC. Only a single line should need to be changed if your Makefile was properly designed, although some tuning might be needed in `main.c`. Demonstrate that the execution of the resulting program on the PC leads to the correct result.
8. Modify your `Makefile` to compile the program using fixed point calculation targeted towards the STM32 microcontroller.
9. Execute in the Qemu emulator and demonstrate that the correct result is displayed, matching the fixed point calculation on the PC and the floating point calculation.

We have filled the initial state of the calculation with a velocity information whereas the on-board sensor is an accelerometer. The following chart is cropped from [2]. Based on your previous software and on the information provided in this chart,

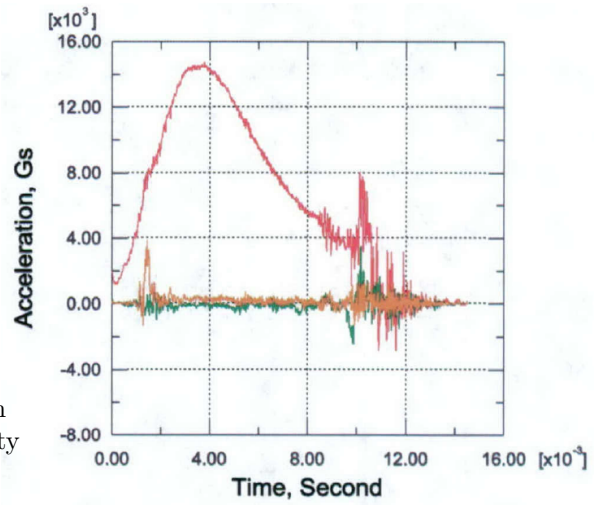

Figure 2: Shell acceleration as a function of time [2]

10. how can you deduce roughly the exit velocity of the shell ejected from the gun? Remember that 1 G is 10 m.s^{-2}? How do you approximate the provided curve to assess whether the result is reasonable?
11. repeat question 3 or 9 with this new input, replacing the velocity information with an acceleration information. How do the results compare?
12. We have so far considered the ideal case of no drag: viscous air friction creates a negative acceleration proportional to the square of the velocity $a_x = -K \cdot v^2$ with the proportionality coefficient $K = \frac{1}{2}\rho C_d A/m$ with $A = \pi D^2$ the exposed area of the shell of diameter $D = 0.155$ m,

$C_d = 0.1$ the unit-less drag coefficient and $\rho = 1.25$ k.m^{-3} the density of air and $m = 43$ kg the mass of the shell. This numerical application leads to $K = 10^{-4}$. Update the shell path model accordingly, assuming that the drag only acts on the horizontal (x) component of the velocity, meaning that the acceleration along the X axis might no longer be constant as was the case earlier. Start with the floating point implementation before addressing the fixed point approach, which might require a different scaling factor than used previously. How did the range evolve according to this drag correction?

Questions (all subquestions valid otherwise answer is void):

13. Select the 13th bit of a 16-bit variable in a `if` condition: how would you define such a variable in C and how will you test its 13th-bit state?
14. How would you set to one the 3rd bit of an 8-bit variable, and how is this variable defined in C?
15. What is the `gcc` option to stop the compilation at the object (.o) generation?
16. How would you tell `gcc` to search for header files in the `/home/xxx/include` directory? Which compilation step uses this flag in the full compiler sequence?
17. How would you tell `gcc` to search for libraries in the `/home/xxx/lib` directory? Which compilation step uses this flag?
18. How would you tell `gcc` to link the `libmyfunc.a` library to generate the executable?
19. How can we read the 8-bit content of the memory address `addr` in C and store it in a variable `v`? how is `v` defined?
20. What variable definition prefix in C allows for allocating memory on the heap rather than on the stack?
21. What is the heap location in RAM whem the processor just booted?
22. What is the stack location in RAM after initializing the processor registers?

Copying stupidly Google's off-topic answer will lead to negative grades.

References

[1] R. Lee, *Instrumented projectile firings in a 155-mm regenerative liquid propellant gun (RLPG) system*, Technical Report AD-B177 784 (1993) found at `https://apps.dtic.mil/sti/pdfs/ADB177784.pdf`

[2] H. Cordes *& al.*, *Design Accelerations for the Army's Excalibur Projectile*, Technical Report ARAET-TR-05008 (2005) found at `https://www.researchgate.net/publication/235199081_Design_Accelerations_for_the_Army%27s_Excalibur_Projectile`

Solutions

1.
```
struct shell_struct {float ax, float ay, float vx, float vy, float x, float y;} shell;
```

2.
```
struct shell_struct next_time(struct shell_struc s,float dt)
{ struct shell_struct tmp;
  tmp.ax=s.ax;          tmp.ay=s.ay;
  tmp.vx=s.vx+s.ax*dt; tmp.vy=s.vy+s.ay*dt;
  tmp.x =s.x+tmp.vx*dt;tmp.y =s.y+ tmp.vy*dt;
}
```

3.
```
#include "shell_float.h"
#include <math.h>
#include <stdio.h>

int main()
{struct shell myshell;
 float dt=0.001;  // 1 ms
 myshell.ax=0;   myshell.ay=-10;
 myshell.vx=1000*cos(45./180.*M_PI);
 myshell.vy=1000*sin(45./180.*M_PI);
 myshell.x=0;    myshell.y=0;
 do {myshell=iterate(myshell,dt);printf("%f\n",myshell.x);} while (myshell.y>=0);
}
```

4.
```
PROGRAMME=shell_float

all: shell

shell: main.o $(PROGRAMME).o
gcc -o shell main.o $(PROGRAMME).o

$(PROGRAMME).o: $(PROGRAMME).c
gcc -c $(PROGRAMME).c

main.o: main.c
gcc -c main.c
```

5. The shell falls back on the ground when its vertical position y is null. The range under the current conditions is the x position when this condition is met: 100 km.

6. 3 decimals is a scale factor of $1000 \simeq 2^{10}$ so that 10 bits are used in the integer representation of the fixed point arithmetic.

7. `PROGRAMME=shell_fixed` in the Makefile

8. the compiler becomes `arm-none-eabi-gcc` with the appropriate options to compile on the Cortex M3.

9.
```
#include "shell_fixed.h"

long addfix(long in1,long in2) {return(in1+in2);}

long mulfix(long in1,long in2)
{long long tmp;
 tmp=(long long)in1*(long long)in2; tmp/=SCALE;
 return((int)tmp);
}

long divfix(long in1,long in2)
{long long tmp=(long long)in1*SCALE;
 if (in2!=0) tmp/=(long long)in2; else tmp=0;
 return((long)tmp);
}

struct shell iterate(struct shell in,long dt)
{struct shell out;
 out.ax=in.ax;                          out.ay=in.ay;
 out.vx=addfix(in.vx,mulfix(out.ax,dt));out.vy=addfix(in.vy,mulfix(out.ay,dt));
 out.x=addfix(in.x,mulfix(out.vx,dt));  out.y=addfix(in.y,mulfix(out.vy,dt));
```

```
 return(out);
}
```

with the header file

```
#define mytype long
#define SCALE  1000
struct shell {long ax;long ay;long vx;long vy;long x;long y;};
struct shell iterate(struct shell, long);
```

and the `main.c` file becoming

```
#include "shell_fixed.h"
#include <math.h>
#include <stdio.h>

int main()
{struct shell myshell;
 mytype dt=(mytype)(0.001*SCALE);
 myshell.ax=0;
 myshell.ay=(mytype)(-10*SCALE);
 myshell.vx=(mytype)(1000*cos(45./180.*M_PI)*SCALE);
 myshell.vy=(mytype)(1000*sin(45./180.*M_PI)*SCALE);
...
```

the result closely matches the floating point calculation with a result of 99984647 divided by the scale factor of 1000 or 99.98 km

10. the mean acceleration is about 8000 G during about 5 ms so that the muzzle velocity is $8 \times 5 \times 10$ m/s. This velocity of 400 m/s is in the lower range of published results and seems appropriate.

11. in the numerical approximation with 1 ms step, we can iterate the first steps of the simulation with an acceleration found on the chart, namely

time (ms)	acceleration ($\times$1000G)
0	1
1	4
2	8
3	14
4	14
5	12
6	9
7	7
8	6
9	5
10	4

These values are input in an array fed during the first iterations of the trajectory calculation instead of the initial velocity.

12.
```
struct shell iterate(struct shell in,float dt)
{float K=1e-4;
 struct shell out;
 out.ax=-K*in.vx*in.vx;      out.ay=in.ay;
...
```

Using the same initial conditions of the fixed velocity of 1000 m/s at an angle of 45°, the range drops to 24 km due to air drag.

13. `short s;if (((s>>13)&1)!=0) ...`

14. `char c;c|=(1<<3);`

15. `gcc -c`

16. `-I` is needed by the preprocessor

17. `-L` is needed by the linker

18. `-ltoto`

19. `char v=*(char*)addr;`

20. `static`
21. at the beginning of the address space allocated to RAM
22. at the end of the address space allocated to RAM