DS4 Drop LAB REPORT

LAB #5

SECTION M

SUBMITTED BY:

SHOUNAK LAHIRI

SUBMISSION DATE:

10/17/2018

Part 1:

Problem

The purpose of this lab is to create a program that will be able to detect when the DualShock 4 controller is in free fall, while it narrates what is happening to it, printing periods when the controller is not falling and printing exclamation points when the controller is in free fall; additionally, printing the distance the controller fell as well as the time it took to fall. To help write the program, the program should use the previously written mag function, which calculates the magnitude of acceleration, and the closeTo function which returns a 1 if the number and point provided are within the tolerance value which is also provided.

Analysis

The only way to be able to detect when the controller is falling is to put the code for checking for free fall inside a while loop that should always run, this loop would be used while the controller is not in free fall. Another while loop should be used when the controller is in free fall. Additionally, there should be multiple scan statements in the code, to allow for checking the conditions with current data. By implementing multiple scan statements throughout the code, it ensures that when the code is running new information from the controller is being taken into account which should give the program a better "response" time.

Design

Our problem was to output distance and time the controller took to fall. This task could be broken into smaller problems/steps.

1. Create a while loop for when the controller is not falling
2. Create a while loop for when the controller is falling
3. Calculate the time and distance that the controller fell
4. Output the calculated values

To start we needed to create a loop to wait for the controller to be in free fall and print periods during that time. Using the close_to and mag functions we were able to find when the controller was in free fall. When the close_to function returned a 1 for the magnitude of acceleration being close to 1 it meant that the controller was not falling. While the controller was in this state, we needed to print periods which was achieved with a simple printf statement.

Next, we needed to create a loop for when the controller was in free fall. By knowing when the controller was not falling, we also learned when the controller was in free fall (close_to function returned a 0 for the magnitude of acceleration being close to 1). While the controller was in free fall we needed to print "Help me, I'm Falling" once, and exclamation points the entire time the controller is falling, which was achieved with a simple printf statement.

After figuring out when the controller was falling and when it was not, we needed to calculate the distance that the controller fell and the time it took to fall. We started with the easier of the two, the time. To find the time it took the controller to fall we created two variables, one which would hold the start time from the first scan, the other one would hold the end time from another scan statement. To calculate the time it took for the controller to fall we simply subtracted the start time from the end time. Calculating the distance was a little more difficult, we had to use the formula which was provided, but we needed to calculate the velocity in order to use it. To calculate the velocity of the fall we used the mag function with the values for the acceleration.

Finally, we had finished the while loops and calculated the values of the distance fallen and the time taken for the controller to fall. All that was left to do was to print out the results. We achieved this with a simple printf statement. The harder part of this process was figuring out where to put the statement because we only wanted it to print once, at the end of the fall/program.

Testing

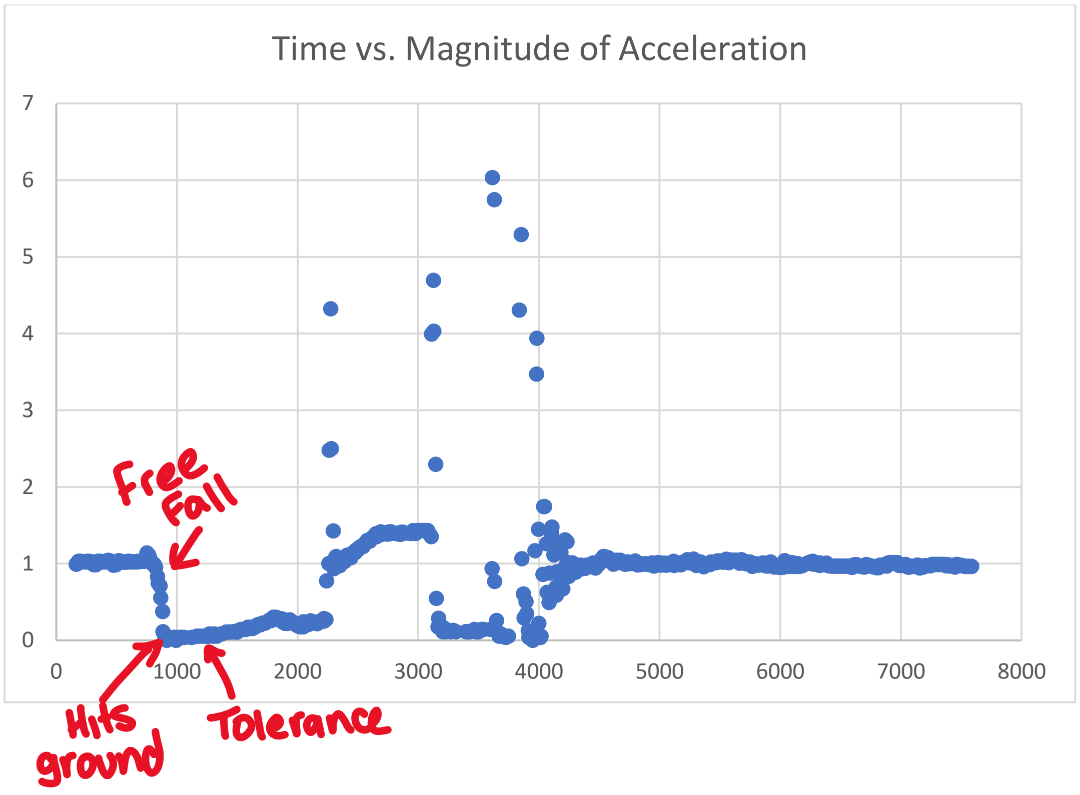
To test that the program was working correctly, we checked the program at various steps by using printf statement like breakpoints, to help see what the program was doing and the results that it was calculating. When we got the entire program working how we thought it should, we used the sample data to test the program. Once we got the output that the TA’s had shown us for the sample data set we knew that we had a working program that was giving accurate results.

Comments

I think the design that was implemented was pretty good at doing everything the program had to do in a timely manner. I learned that where you put printf and scanf statements is very important especially when working with loops that run almost forever. Additionally, I found that the adjusting the tolerance level in the close_to function can greatly alter the results of the program.

Questions/Experiments:

1. The results are fairly consistent, they all show a fallen distance less than one meter. The results may vary because the place where we were dropping/catching the controller was not the same each trial.
2. The program says that the distance the controller fell was 8.46 meters.
3. (Next Page)


Time vs. Magnitude of Acceleration
0
1
2
3
4
5
6
7
0
1000
2000
3000
4000
5000
6000
7000
8000
Free Fall
Hits ground
Tolerance

Part 2

Problem:

Although the program is outputting a distance that the controller falls, it is not providing an accurate fall distance, our calculations assume that we live in a world with out opposing forces. To fix this problem we need to calculate the fall distance while taking into account air resistance on the controller. Additionally, the program should be able to calculate how much less this new distance, that take air resistance into account, is than the original calculated fall distance.

Analysis

To take air resistance into account when calculating the distance that the controller falls requires the use of the principle of integrals to find the distance the controller falls at each velocity that the controller reaches while it is falling. To calculate the percent that the distance with air resistance taken into account is less than the original fall distance, simply divide the air resistance fall distance by the original fall distance, then subtract that number from 1 and multiply the result by 100.

Design

Our problem to calculate the fall distance while taking air resistance into account and printing out the value how much less that distance is over the original fall distance can be broken into two problems/steps:

1. Calculate and print the air resistance
2. Calculate and print the percent that the air resistance fall distance was less than the original fall distance

To find the air resistance we have to use the fact that the velocity of the falling controller can be used to find the distance that the controller fell by multiplying it by the time and adding them up which results in the distance that the controller falls. Then using a simple printf statement outside of the loop along with the statement in the instruction manual we were able to print the resulting distance that takes the air resistance into account.

Calculating the percent of how much less the fall distance with air resistance taken into account vs the original fall distance is much simpler than actually calculating the distance while taking the air resistance into account. To find the percent, we have to divide the distance with air resistance taken into account by the original fall distance. Take the result and subtract it from 1 to get the decimal value of how much less the air resistance distance is, multiply this number by one hundred to get the percent value. Using another printf statement, we were able to print the value of the percent, along with some lines of text that were in the instruction manual.

Testing

To test that the program was working correctly we used the sample data that was provided to us in the lab. The TA’s told us what the correct distance should be when the air resistance is taken into account. We tested the program by running the first sample data file called sampledata2013_1.csv through the program to get the results. We continued testing with this method until we go the correct result.

Comments

I think the design that was implemented was pretty good at doing everything the program had to do in a timely manner. I learned that a lot of the notations that we use in math do not apply when programming. For example, even when numbers are inside parenthesis, they do not automatically multiply there has to be an asterisk to show that they multiply.

Questions/Experiments

1. When testing in the lab area the difference was usually less than ten percent, when dropping form the second floor the difference was around ten percent.
2. When accounting for the air resistance the distance from the second floor is about 8.5 meters.
3. The main issue that arose when trying to implement part 2 was not knowing where to calculate what, for instance, should things be calculated inside the while loop or after the loop was done executing.

Source Code

```
#include <stdio.h>
#include <math.h>
#include <unistd.h>

double mag( double x, double y, double z);
int close_to (double tolerance, double point, double value);
int timePrinter (int t);
#define g 9.8

int main()
{
     int t, b1, b2, b3, b4;
    double ax, ay, az, gx, gy, gz, startTime, endTime, timeFall;
     scanf("%d, %lf, %lf, %lf, %lf, %lf, %lf, %d, %d, %d, %d", &t,
&ax, &ay, &az, &gx, &gy, &gz, &b1, &b2, &b3, &b4 );

     printf("Shounak Lahiri\n");
     printf("sbalhiri\n");

     printf("Ok, now I am receiving data\n");
     printf("I'm waiting");


           scanf("%d, %lf, %lf, %lf, %lf, %lf, %lf, %d, %d, %d, %d",
&t, &ax, &ay, &az, &gx, &gy, &gz, &b1, &b2, &b3, &b4 );
           while(close_to(.25, 1, mag(ax,ay,az)) == 1)
           {
                 scanf("%d, %lf, %lf, %lf, %lf, %lf, %lf, %d, %d, %d,
%d", &t, &ax, &ay, &az, &gx, &gy, &gz, &b1, &b2, &b3, &b4 );
                 printf(".");
                 fflush(stdout);
           }

           startTime = t;
           int fallState;
           double total;
           double vel;
           double fallTime;
```

```
            double seconds;
            int loopStart;
            int loopEnd;

            while(close_to(.25, 1, mag(ax,ay,az)) == 0)
            {
                  loopStart = t;
                  scanf("%d, %lf, %lf, %lf, %lf, %lf, %lf, %d, %d, %d,
%d", &t, &ax, &ay, &az, &gx, &gy, &gz, &b1, &b2, &b3, &b4 );
                  if(fallState == 0)
                  {
                        printf("\n Help me, I'm Falling");
                        fallState = 1;
                  }
                  else
                  {
                        printf("!");
                  }

                  loopEnd = t;
                  fallTime = loopEnd - loopStart;
                  seconds = fallTime/ 1000.00;
                  vel  += (g) * (1-mag(ax,ay,az)) * (seconds);
                  total += vel * seconds;
            }

            printf("\n");
            endTime = t;

            double TimeFallen = (endTime - startTime)/1000;
            double distance = (0.5 * g * TimeFallen * TimeFallen);
            double percentLess = (1 - (total/distance)) * 100;
            printf("Ouch! I fell %lf meters in %lf seconds.\n",
distance , TimeFallen);
            printf("Compensating for air resistance, the fall distance
was %lf\n" , total);
            printf("This is %.2lf", percentLess);
            printf("%% than before.");
            fflush(stdout);
```

```c
}

double mag(double x, double y, double z)
{
	double value= pow(x,2) + pow(y,2) + pow(z,2);
	return sqrt(value);
}

int close_to(double tolerance , double point, double value)
{
	double difference;
	difference = fabs(point - value);
	if (difference <= tolerance)
	{
		return 1;
	}
	else
	{
		return 0;
	}

}

int timePrinter(int t)
{
	if(t%1000 == 0)
	{
		return 1;
	}
	else
	{
		return 0;
	}

}
```