DS4 DROP

LAB 5 SECTION 1

SUBMITTED BY:

ETHAN GRIESMAN

SUBMISSION DATE:

10/10/2022

Lab Problem (Part 1)

The purpose of this lab is to familiarize ourselves with if, for, and while statements within the C programming language and employ them to calculate more detailed information regarding the motion of a DualShock 4 controller

Analysis

The first part of the lab instructed us to firstly transfer our magnitude and close_to functions from the previous labs. These will be used to calculate the magnitude of the acceleration along x, y, and z, as well as determine if said magnitude is within a specified range (tolerance) of the target value, respectively. By utilizing both functions in this way, we were to ultimately output information indicating that the controller was falling

Design

Using the basic outline above, I designed the program to print a statement notifying the user that data was ready to be inputted. To accomplish this, I wrote an if statement within a while statement that essentially says that, while the acceleration of the DualShock controller is within 0.25 of the target value of 0 (no acceleration), then take in the input, and increment the time counter.

```
printf("<Ethan><Griesman>\n");
printf("<egriesma@iastate.edu>\n");

scanf("%d, %lf, %lf, %lf", &t, &ax, &ay, &az);

printf("Ok, I'm now receiving data.\n");
printf("I'm Waiting.");
fflush(stdout);
    while (close_to(0.25, 0, mag(ax, ay, az))) {
        scanf("%d, %lf, %lf, %lf", &t, &ax, &ay, &az);
        timeCount = timeCount++;
```

From here, I wrote an if statement stating that, if the current time, defined as “t” becomes greater than or equal to timeCount+25, print out a period (.), indicating to the user that the program is waiting.

```
if (t >= timeCount+10) {
    printf(".");
    timeCount = 0;
    fflush(stdout);
}
```

```
egriesma@CO2048-06 /cygdrive/u/CPR_E_185/LAB5
$ ./ds4rd.exe -d 054c:05c4 -D DS4_USB -t -g | ./lab5_first_demo.exe
<Ethan><Griesman>
<egriesma@iastate.edu>
Ok, I'm now receiving data.
I'm Waiting.....................................................
```

After this, I declared dropTime, an int I defined earlier, to be equal to “t”. This basically redefines “t” to now measure the time the controller is in freefall. Once magnitude of the acceleration of the controller changes (i.e, it is dropped from a certain height), I wrote a printf statement to display to the user that the controller was falling. Similar to what was done in the previous step, I wrote a while statement to state that, while the magnitude of the acceleration is less than or equal to an arbitrary value of 0.8, increment the time, and output exclamation points (!) to indicate that the controller is indeed in freefall.

```
egriesma@CO2048-06 /cygdrive/u/CPR_E_185/LAB5
$ ./ds4rd.exe -d 054c:05c4 -D DS4_USB -t -g | ./lab5_first_demo.exe
<Ethan><Griesman>
<egriesma@iastate.edu>
Ok, I'm now receiving data.
I'm Waiting.....
Help me! I'm falling!!!!!!!!
```

Testing

Unlike other labs, a lot of testing had to be done to refine the code for this program. Testing different tolerances to make sure the program was too sensitive, and making sure I defined time properly to make the right measurements were the biggest initial challenges for the first part of the lab. By changing the tolerances, as well as declaring time in the right locations, I managed to get a fairly reasonable time and distance measurement.

RESULT:

```
egriesma@CO2048-06 /cygdrive/u/CPR_E_185/LAB5
$ ./ds4rd.exe -d 054c:05c4 -D DS4_USB -t -g | ./lab5_first_demo.exe
<Ethan><Griesman>
<egriesma@iastate.edu>
Ok, I'm now receiving data.
I'm Waiting.....
Help me! I'm falling!!!!!!!!
Ouch! After 0.530 seconds, I fell 1.376 meters.
```

Comments

In this lab, the biggest lesson I learned was how to properly format if and while statements. Initially I was having trouble getting the time delay right until I was able to

enter my scanf inside of the if statement, as well as timing the “I’m falling” printf statement correctly to display only when the controller has started falling. All in all, it really boiled down to simple trial and error.

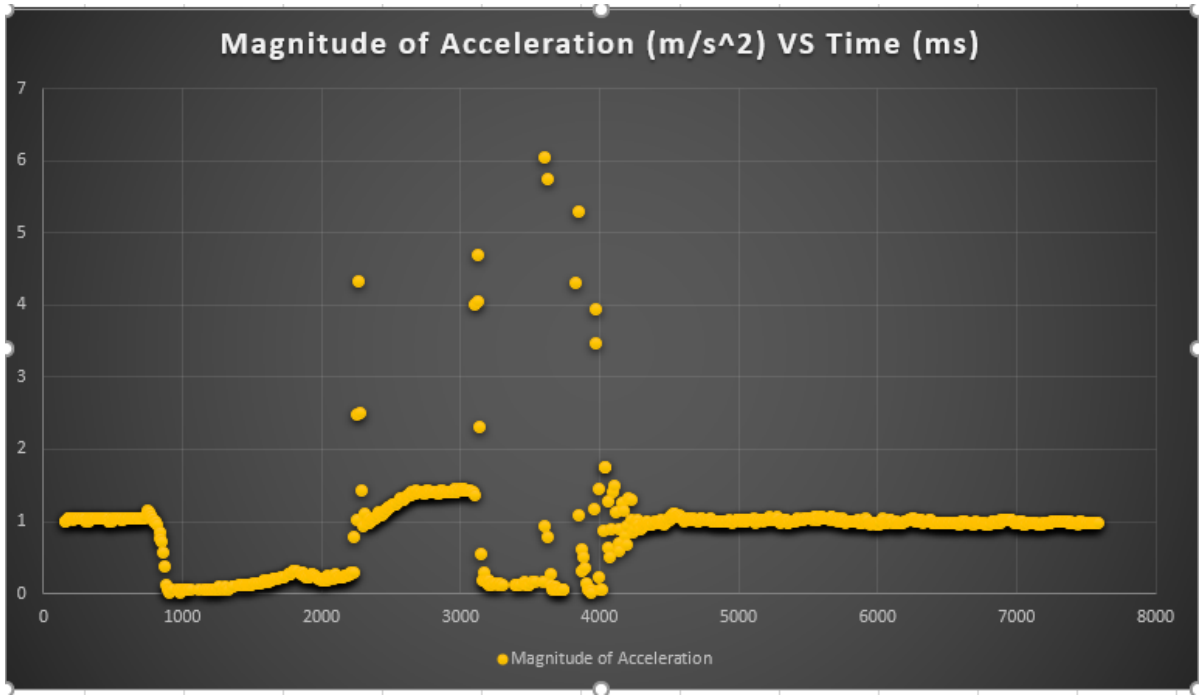
Question 1:

At first, the times, and not the distances, were the most consistent. This puzzled me at first, until I realized that I had not used the proper freefall formula (½)gt^2 to calculate the distance the controller had fallen. Once I changed this, readjusted my tolerances a bit, and did some troubleshooting, I was able to get distances that were both consistent and reasonable. Another reason for variation I believe was restarting my code while I was still lifting the controller up from the previous test and not while it was at rest, which is the first condition that must be met in the program.

Question 2:

According to my program, when running the sample data, the distance is around 11 meters or roughly 36 feet. I got varying results from the different sample programs using the same code, but this was the one that seemed most reasonable.

Question 3



Viewing the graph, it can be seen that the magnitude of the acceleration of the DualShock controller is fairly constant from t=0 to t is roughly 3000. From this, it can be assumed that the controller is in freefall from t=3250 to t=4350, hitting the ground resulting in again a fairly constant, non changing acceleration magnitude. This was using tolerances of 0.25 and 0.05, to start and end the program respectively. My belief is that

the reason the magnitude is 1 initially, and then 0, and then 1 again once the controller has called could be due to a tolerance that is much closer to zero than what would be ideal. Put another way, once the controller hits the ground, it doesn't stop moving completely. It will hit the ground and then move slightly before coming fully to rest, this movement immediately before the controller comes full to rest is what likely caused the magnitude to be 1 after the fall instead of the expected value of zero.

Lab Problem (Part 2)

The overall purpose of the second part of this two week lab is to, using the code from the previous week, familiarize ourselves with programs using multiple while and if statements to output properties regarding the freefall motion of a DualShock 4 controller. We are particularly going to factor in the effects of air resistance as well as calculate percent difference.

Analysis

For the second part of the lab, our instructions were to transfer all of the data over from the first week, but now with the added objective of factoring in air resistance as well as an error measurement to compare the current distance to the previous one. To accomplish this, I first declared an explicit value for the distance fallen called displacement (disp), setting it equal to 0 (since initially, there is no displacement).

Design

Using the basic outline above, I renamed my distanceFallen parameter to distance, and defined a velocityPrev to measure the previous velocity.

```
int main(void) {
    int t;
    double ax, ay, az;
    int timeCount = 0;
    double distance;
    double endTime;
    int dropTime;
    double velocityPrev = 0;
    double prevTime = 0;
    double disp = 0;
    double percent = ((distance - disp) / distance) * 100.0;
    double velocity;
```

From here, the code runs identically to the previous step. Initially, the program begins, printing a statement to the user indicating that data may not be taken in, and then after a time delay tells the user "I'm Waiting" followed by a period. Then, when the controller is dropped, the program again prints "I'm Falling" The difference, though, between this part and the previous part is that I then defined explicitly my velocity as instructed by the

formula

```
velocity += 9.8 * (1-mag(ax, ay, az)) * (t/1000.0 - prevTime/1000.0);
```

This defines the velocity as changing according to the magnitude of the acceleration, the initial time, and the previous time. This is only true once the controller begins its descent. I then declared explicitly my displacement, which is required for the comparison in distances between each fall. My time is divided by 1000 to convert from ms to seconds.

```
disp += velocity * ((t-prevTime)/1000.0);
```

Now that I've defined my velocity and displacement, which are being calculated at the same time as my program is designed to output (!) after the "Help! I'm Falling" statement, I want to define an end time (in seconds), my distance (same code as before), and then the percent difference between the distance fallen, and the displacement from the previous fall. I then want to output these as statements telling the user how my program is accounting for air resistance, and the percent difference between the current fall and the last one.

```
endTime = (t-dropTime) / 1000.0;

distance = (0.5) * (9.8) * (endTime * endTime);
percent = ((distance - disp) / distance) * 100.0;

printf("\nOuch! After %4.3lf seconds, I fell %4.3lf meters.", endTime, distance);

printf("\nCompensating for air resistance, that fall was %4.3lf meters.", disp);

printf("\nThis is %2.0lf%% less than computed before.", percent);
urn 0;
```

RESULT:
Demo'd to TA - 10/5/2022

Testing

While the first part of this 2 week lab took a lot of testing to obtain accuracy, this part took an even greater amount of effort and troubleshooting. At first, I was using an incorrect formula to calculate air resistance using previous velocity that resulted in very unreasonable falling distances. Then, after modifying my code and testing a few more times, I was able to get results that made a lot more sense.

Comments

All in all, my greatest mistake from this lab was in how I calculated my air resistance. I was neglecting to use the gravitational acceleration and subtract the times. Also, looking back at my code for this step, I noticed that my velocityPrev double is actually redundant, as there was no need to use it to obtain the desired results. In the future, I think I could definitely spend a little more time trying to create a more concise, easy to read calculation for the air resistance.

Question 1:

The differences were much greater between successive drop tests within the lab than from the 2nd floor. This was mainly due to varying tolerances that were often too small as well as the naturally smaller distances we were dropping the controller from. With the actual drop from the 2nd floor, this wasn't an issue, which resulted in a much more accurate measurement because the distance was so much larger than the few meters we could drop it from in the lab.

Question 2:

Running the sample code with my program, I got a measured distance of around 5 meters fallen in 1.506 seconds, which is roughly 16 feet.

Question 3:

The main difficulties that arose in implementing Part 2 were the discrepancies in values I got from the sample data being run through my program. Most of this I believe was due to me not correctly compiling, or perhaps even having tolerances that were too close to zero.

```
-                                       SE/CprE 185 Lab 04
-              Developed for 185-Rursch by T.Tran and K.Wang
Ethan Griesman
CPR E 185 Section 6
9/25/2022

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

int close_to(double tolerance, double point, double value);
double mag(double x, double y, double z);

int main(void) {
    int t;
    double ax, ay, az;
     int timeCount = 0;
     double distanceFallen;
     double endTime;
     int dropTime;


     printf("<Ethan><Griesman>\n");
     printf("<egriesma@iastate.edu>\n");

    scanf("%d, %lf, %lf, %lf", &t, &ax, &ay, &az);

     printf("Ok, I'm now receiving data.\n");
     printf("I'm Waiting.");
     fflush(stdout);
           while (close_to(0.25, 1, mag(ax, ay, az))) {
                 scanf("%d, %lf, %lf, %lf", &t, &ax, &ay, &az);
                 timeCount = timeCount++;

                       if (t >= timeCount+10) {
                             printf(".");
                             timeCount = 0;
                             fflush(stdout);
                       }
           }
           timeCount = t;
```

```
	dropTime = t;
	printf("\nHelp me! I'm falling");
	fflush(stdout);

		while (mag(ax, ay, az) <= 0.8) {
			timeCount = timeCount++;

				if (t >= timeCount+25) {
					printf("!");
					timeCount = t;
					fflush(stdout);
				}
			scanf("%d, %lf, %lf, %lf", &t, &ax, &ay, &az);
				if(close_to(0.05, 0, mag(ax, ay, az))) {
					break;
				}
		}
		endTime = ((t-dropTime) / (1000.0));
		distanceFallen = 0.5 * 9.8 * (endTime * endTime);

		printf("\nOuch! After %4.3lf seconds, I fell %4.3lf meters.",
endTime, distanceFallen);
	return 0;
}



/* Put your functions here */
double mag(double x, double y, double z) {
double magg = sqrt((x*x)+(y*y)+(z*z));
	return magg;
}
int close_to(double tolerance, double point, double value) {
	if (fabs(point-value) < tolerance) {
		return 1;
	}

	return 0;
```

```
/*--------------------------------------------------------------------
----
-                                        SE/CprE 185 Lab 04
-             Developed for 185-Rursch by T.Tran and K.Wang
Ethan Griesman
CPR E 185 Section 6
10/5/2022

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

int close_to(double tolerance, double point, double value);
double mag(double x, double y, double z);

int main(void) {
    int t;
    double ax, ay, az;
      int timeCount = 0;
      double distance;
      double endTime;
      int dropTime;
      double velocityPrev = 0;
      double prevTime = 0;
      double disp = 0;
      double percent = ((distance - disp) / distance) * 100.0;
      double velocity;


      printf("<Ethan><Griesman>\n");
      printf("<egriesma@iastate.edu>\n");

    scanf("%d, %lf, %lf, %lf", &t, &ax, &ay, &az);

      printf("Ok, I'm now receiving data.\n");
      printf("I'm Waiting.");
      fflush(stdout);
            while (close_to(0.05, 1, mag(ax, ay, az))) {
                  scanf("%d, %lf, %lf, %lf", &t, &ax, &ay, &az);
                  timeCount = timeCount + 1;
```

```
                        if (timeCount == 10) {
                              printf(".");
                              timeCount = 0;
                              fflush(stdout);
                        }
            }

      dropTime = t;
      prevTime = t;
      printf("\nHelp me! I'm falling");
      fflush(stdout);
            while (mag(ax, ay, az) < 1.25) {
                  scanf("%d, %lf, %lf, %lf", &t, &ax, &ay, &az);
                  timeCount++;

                  velocity += 9.8 * (1-mag(ax, ay, az)) * (t/1000.0 -
prevTime/1000.0);

                  disp += velocity * ((t-prevTime)/1000.0);

                  prevTime = t;

                        if (timeCount == 10) {
                              printf("!");
                              timeCount = 0;
                              fflush(stdout);
                        }
                        fflush(stdout);
                        }

            endTime = (t-dropTime) / 1000.0;

            distance = (0.5) * (9.8) * (endTime * endTime);
            percent = ((distance - disp) / distance) * 100.0;

            printf("\nOuch! After %4.3lf seconds, I fell %4.3lf meters.",
endTime, distance);

            printf("\nCompensating for air resistance, that fall was
%4.3lf meters.", disp);

            printf("\nThis is %2.0lf%% less than computed before.",
percent);
      return 0;
}


/* Put your functions here */
double mag(double x, double y, double z) {
double magg = sqrt((x*x)+(y*y)+(z*z));
      return magg;
}
int close_to(double tolerance, double point, double value) {
      if (fabs(point-value) < tolerance) {
```

```
            return 1;
        }

        return 0;

}
```