**LAB REPORT**

**LAB 6**

**SECTION A**

**SUBMITTED BY:**

**WAN ZULSARHAN WAN SHAARI**

**SUBMISSION DATE:**

**3/12/2014**

**Lab Problem**

This lab is called ExDrop and it is divided into two parts. In part 1, basically we have to develop an application to detect free fall and narate the action. In part 1 we only have to drop the espora into our partners hands (about 1 meter) and simulate the programs using the last year data of free fall. Meanwhile in part 2, we will be using a wireless Esplora and drop it off of the 3rd floor balcony in Coover. And we also have to modified the program so that it takes account the effect of air resistance when calculating the distance of falling. The objective of this lab is to develop problem solving skills and develop the skills in using numerous loops to solve a problem.

**Analysis**

For this lab we are going to need some method from the previous lab such as

double mag(double ax, double ay, double az) and int close\_to(double tolerance, double point, double value).

For the first part of the lab we are going to use this equation : x1 = x0 + vt + (1/2)gt2to calculate the distance of the free fall. Where v is the initial velocity (0) , t is the fall time, x1 is the location before, x0 is the location after the fall and g is the acceleration due to graviti (9.8 m/sec2)

Our output should looks like :

**Ok. I’m now receiving data.**

**I’m Waiting ............**

**Help me! I’m falling!!!!!!!!!!!!**

**Ouch! I fell 1.633 meters in**

**0.577 seconds.**

For the second part of our lab, we need to calculate the distance of free fall but this time we are considering the effect of air resistance. To do that, we are going to use the following formula :

vi = vi – 1 + (g – mag(acc)) (ti – ti – 1)

xi = xi – 1 + vi (ti – ti – 1)

The xi should be the distance of the free fall when the Esplora stops falling. After the output from part 1, the program should output :

**Compesanting for air resistance, the fall was 8.777 meters.**

**This is 7% less than computed before.**

**Design**

For the first part of the lab, to calculate the distance of the free fall, we implement a function named calculatedDistance that takes the time taken for the Esplora to hit the ground as the argument then we return the distance stored in variable of type double. The equation that we use is:

distance = (1.0/2) \* GRAV \* timeFall \* timeFall;

GRAV is a constant with value 9.8 which represent the acceleration due to gravity.

Meanwhile, for the second part of the lab, we add two new functions to calculate velocity and also another fuction to calculate distance based on velocity. The function prototypes look like as shown below:

double calculateVelocity(double vOld,double magnitude, int tOld, int tNew);

double calculateDist(double vNew,double dOld, int tOld, int tNew);

We call both of these functions and calculate the velocity and distance as the Esplora is falling. By calculating the distance based on velocity, we are actually considering the effect of the air resistance on the Esplora. When the Esplora hit the ground, the last distance that was capture is the actual distance of the free fall with air resistance. At the of the program we also calculate the percentage different of distance with the air resistance and without air resistance. But issue that arose when we implementing the Part 2 of the lab is that the distance that we get when we considered the air resistance is bigger when we neglected the air resistance. They should be the other way around. But we solved this issue by increasing the tolerance value by 0.5.

**Testing**

For the first part of the lab, we test our program by doing 5 times 1 meter drop in the lab. All the results was quite consistent with only not more than 0.5 meter error. The variation only can be caused because we are not considering the air resistance. Then we also test our program by using the data of the free fall from last year. According to our code, the distance of the third floor balcony in Coover to the ground is 9.68 meters which is quite close to the actual value.

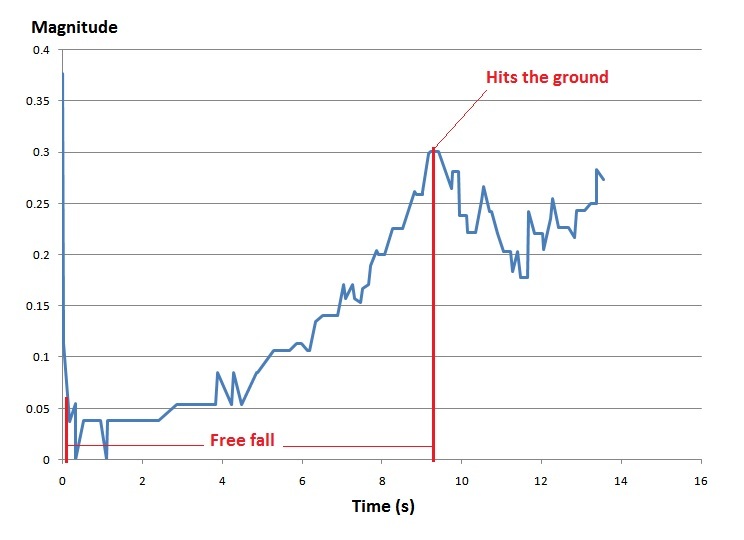
Meanwhile, for the second part of the lab, we still use the same test as the part 1. We try dropping the Esplora in the lab and we record the percentage different between the two distance. Then we drop the Esplora from the third floor and we found that the percentage different between two distance is higher than that of when we do it in the lab. Anyway the distance that we get is quite the same as the distance required by the TA. Our demo was succesfull.

**Comments**

I think the ExDrop lab is the most interesting and challenging lab I had so far. There are a lot of thing that I learned from this lab. The most significant thing is that, we are finally engaging with the outside world when we demo our program outside the lab. It was a new experience for me. All though there are a lot of time and hardship that we spent on this lab but compare to the experince and new skills that I gain, its definitely worthy.

**GRAPH**

Graph of the magnitude of the acceleration as a function of time.



The tolerance value that we use here is 0.4.

**CODES**

**PART 1**

/\* Lab 6

Wan Zulsarhan Wan Shaari - Section A

\*/

#include <stdio.h>

#include <math.h>

#define TRUE 1

#define GRAV 9.8

/////My function prototypes/////

//computes magnitude of acceleration.

double mag(double ax, double ay, double az);

//determines if value is within tolerance of point

int close\_to(double tolerance, double point, double value);

double calculateDistance(double timeFall);

int main(void) {

int t, b1, b2, b3, b4, b5, s;//vars for scanning explore.exe outputs.

double ax, ay, az; //three axis from explore.exe

int reducedOutput = 100;

//My variables for lab5

double magnitude;//current magnitude to be stored here.

int isStill;//boolean for determining if esplora is at rest.

int isFalling = 0;

int upIndex = 0;//0: NONE, 1: TOP, 2: BOTTOM, 3: RIGHT, 4: LEFT

int previousUpIndex = 0;

int startFall\_t;

int endFall\_t;

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

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

magnitude = mag(ax, ay, az);

isStill = close\_to(0.4, 1.05, magnitude);

printf("I'm waiting ");

while(isStill){

//esplora is still as determined by a magnitude close to 1

scanf("%d, %lf, %lf, %lf",

&t, &ax, &ay, &az);

magnitude = mag(ax, ay, az);

//determine if esplora is stationary (1 = true, 0 = false)

isStill = close\_to(0.4, 1.05, magnitude);

if(t%reducedOutput >= 0 && t%reducedOutput < 3){

printf(".");

fflush(stdout);

}

}

// Magnitude was not in range of ~1, so it broke out of still loop

printf("\n\n Help me! I'm falling!");

startFall\_t = t;

double previousMag = magnitude;

while(1){

scanf("%d, %lf, %lf, %lf",

&t, &ax, &ay, &az);

magnitude = mag(ax, ay, az);

//if there is a severe sudden change in magnitude

if(sqrt((previousMag-magnitude)\*(previousMag-magnitude))>0.4)

{

break;

}

if(t%reducedOutput >= 0 && t%reducedOutput < 3){

printf("!");

}

previousMag = magnitude;

}

endFall\_t = t;

double timeFall = (double)(endFall\_t - startFall\_t)/1000;

double distance = calculateDistance(timeFall);

printf("\n\n Ouch! I fell %.3lf meters in %.3lf seconds.", distance,timeFall);

fflush(stdout);

return 0;

}

////////////////////////////////////////////

//Our functions.

double calculateDistance(double timeFall){

double toReturn;

toReturn = (1.0/2) \* GRAV \* timeFall \* timeFall;

return toReturn;

}

/\* Put your lab 4 functions here, as well as your new function close\_to \*/

double mag(double ax, double ay, double az){

double magnitude = sqrt((ax\*ax)+(ay\*ay)+(az\*az));\

return magnitude;

}

//determines if value is within tolerance of point

int close\_to(double tolerance, double point, double value){

int inRange = 0;//default to false

double toleranceLow = point - tolerance;//minimum allowed

double toleranceHigh = point + tolerance;//maximum allowed

if(value >= toleranceLow && value <= toleranceHigh){

inRange = TRUE; //true

}

return inRange;

}

/\* Lab 4 Wrapper Program \*/

#include <stdio.h>

#include <math.h>

#define TRUE 1

/\* Put your function prototypes here \*/

double mag(double ax,double ay,double az);

int minutes(int t);

int seconds(int t);

double millis(int t);

int main(void){

int t;

double ax, ay, az;

while(TRUE){

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

/\* CODE SECTION 0 \*/

double tSeconds =(double) t/1000;

printf("Echoing output: %8.3lf, %7.4lf, %7.4lf, %7.4lf\n", tSeconds, ax, ay, az);

/\* CODE SECTION 1 \*/

printf("At %d ms, the acceleration's magnitude was: %lf\n",

t, mag(ax, ay, az));

/\* CODE SECTION 2\*/

printf("At %d minutes, %d seconds, and %3.0lf milliseconds it was: %lf\n",

minutes(t), seconds(t), millis(t), mag(ax,ay,az));

}

return0;

}

/\* Put your functions here \*/

double mag(double ax,double ay,double az)

{

double toReturn = sqrt((ax\*ax)+(ay\*ay)+(az\*az));\

return toReturn;

}

int minutes(int t){

int toReturn =(t/1000)/60;

if(t%reducedOutput >= 0 && t%reducedOutput < 3){

printf(".");

fflush(stdout);

}

}

// Magnitude was not in range of ~1, so it broke out of still loop

printf("\n\n Help me! I'm falling!");

startFall\_t = t;

double previousMag = magnitude;

while(1){

scanf("%d, %lf, %lf, %lf",

&t, &ax, &ay, &az);

magnitude = mag(ax, ay, az);

//if there is a severe sudden change in magnitude

if(sqrt((previousMag-magnitude)\*(previousMag-magnitude))>0.4)

{

break;

}

if(t%reducedOutput >= 0 && t%reducedOutput < 3){

printf("!");

}

previousMag = magnitude;

}

endFall\_t = t;

double timeFall = (double)(endFall\_t - startFall\_t)/1000;

double distance = calculateDistance(timeFall);

printf("\n\n Ouch! I fell %.3lf meters in %.3lf seconds.", distance,timeFall);

fflush(stdout);

return 0;

}

////////////////////////////////////////////

//Our functions.

double calculateDistance(double timeFall){

double toReturn;

toReturn = (1.0/2) \* GRAV \* timeFall \* timeFall;

return toReturn;

}

/\* Put your lab 4 functions here, as well as your new function close\_to \*/

double mag(double ax, double ay, double az){

double magnitude = sqrt((ax\*ax)+(ay\*ay)+(az\*az));\

return magnitude;

}

//determines if value is within tolerance of point

int close\_to(double tolerance, double point, double value){

int inRange = 0;//default to false

double toleranceLow = point - tolerance;//minimum allowed

double toleranceHigh = point + tolerance;//maximum allowed

if(value >= toleranceLow && value <= toleranceHigh){

inRange = TRUE; //true

}

return inRange;

}

////////////////////////////////////////////

//Our functions.

double calculateDistance(double timeFall){

double toReturn;

toReturn = (1.0/2) \* GRAV \* timeFall \* timeFall;

return toReturn;

}

/\* Put your lab 4 functions here, as well as your new function close\_to \*/

double mag(double ax, double ay, double az){

double magnitude = sqrt((ax\*ax)+(ay\*ay)+(az\*az));\

return magnitude;

}

//determines if value is within tolerance of point

int close\_to(double tolerance, double point, double value){

int inRange = 0;//default to false

double toleranceLow = point - tolerance;//minimum allowed

double toleranceHigh = point + tolerance;//maximum allowed

if(value >= toleranceLow && value <= toleranceHigh){

inRange = TRUE; //true

}

return inRange;

}

**PART 2**

/\* Lab 6

Wan Zulsarhan Wan Shaari - Section A

\*/

#include <stdio.h>

#include <math.h>

#define TRUE 1

#define GRAV 9.8

/////My function prototypes/////

//computes magnitude of acceleration.

double mag(double ax, double ay, double az);

//determines if value is within tolerance of point

int close\_to(double tolerance, double point, double value);

double calculateDistance(double timeFall);

double calculateVelocity(double vOld,double magnitude, int tOld, int tNew);

double calculateDist(double vNew,double dOld, int tOld, int tNew);

int main(void) {

int t, b1, b2, b3, b4, b5, s;//vars for scanning explore.exe outputs.

double ax, ay, az; //three axis from explore.exe

int reducedOutput = 100;

//My variables for lab5

double magnitude;//current magnitude to be stored here.

int isStill;//boolean for determining if esplora is at rest.

int isFalling = 0;

int upIndex = 0;//0: NONE, 1: TOP, 2: BOTTOM, 3: RIGHT, 4: LEFT

int previousUpIndex = 0;

int startFall\_t;

int endFall\_t;

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

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

magnitude = mag(ax, ay, az);

isStill = close\_to(0.4, 1.05, magnitude);

printf("I'm waiting ");

while(isStill){

//esplora is still as determined by a magnitude close to 1

scanf("%d, %lf, %lf, %lf",

&t, &ax, &ay, &az);

magnitude = mag(ax, ay, az);

//determine if esplora is stationary (1 = true, 0 = false)

isStill = close\_to(0.4, 1.05, magnitude);

if(t%reducedOutput >= 0 && t%reducedOutput < 3){

printf(".");

fflush(stdout);

}

}

// Magnitude was not in range of ~1, so it broke out of still loop

printf("\n\n Help me! I'm falling!");

startFall\_t = t;

double previousMag = magnitude;

double vOld = 0.0, vNew = 0.0;

double tOld = t, tNew = t;

double dOld = 0.0, dNew = 0.0;

while(1){

scanf("%d, %lf, %lf, %lf",

&t, &ax, &ay, &az);

tNew = t;

magnitude = mag(ax, ay, az);

vNew = calculateVelocity(vOld, magnitude, tOld, tNew);

dNew = calculateDist(vNew, dOld, tOld, tNew);

vOld = vNew;

tOld = tNew;

dOld = dNew;

//if there is a severe sudden change in magnitude

if(sqrt((previousMag-magnitude)\*(previousMag-magnitude))>0.4)

{

break;

}

//determine if esplora is stationary (1 = true, 0 = false)

//isStill = close\_to(0.4, 1.05, magnitude);

if(t%reducedOutput >= 0 && t%reducedOutput < 3){

printf("!");

}

previousMag = magnitude;

}

endFall\_t = t;

double timeFall = (double)(endFall\_t - startFall\_t)/1000;

double dNoResist = calculateDistance(timeFall);

printf("\n\n Ouch! I fell %.3lf meters in %.3lf seconds.", dNoResist,timeFall);

printf("\n\n Compensating for air resistance, the fall was %.3lf meters.", dNew);

int percentDif = sqrt((dNoResist - dNew)\*(dNoResist - dNew))/dNoResist\*100;

printf("\n\n This is %d%% less than computed before.", percentDif);

fflush(stdout);

return 0;

}

////////////////////////////////////////////

//Our functions.

double calculateDistance(double timeFall){

double toReturn;

toReturn = (1.0/2) \* GRAV \* timeFall \* timeFall;

return toReturn;

}

double calculateVelocity(double vOld,double magnitude, int tOld, int tNew){

double toReturn;

toReturn = vOld + (GRAV - magnitude)\*((double)(tNew - tOld)/1000);

return toReturn;

}

double calculateDist(double vNew,double dOld, int tOld, int tNew){

double toReturn;

toReturn = dOld + vNew\*((double)(tNew - tOld)/1000);

return toReturn;

}

/\* Put your lab 4 functions here, as well as your new function close\_to \*/

double mag(double ax, double ay, double az){

double magnitude = sqrt((ax\*ax)+(ay\*ay)+(az\*az));\

return magnitude;

}

//determines if value is within tolerance of point

int close\_to(double tolerance, double point, double value){

int inRange = 0;//default to false

double toleranceLow = point - tolerance;//minimum allowed

double toleranceHigh = point + tolerance;//maximum allowed

/\*printf("Is %6.4lf between %6.4lf and %6.4lf ?\n",

value, toleranceLow, toleranceHigh);//debug\*/

if(value >= toleranceLow && value <= toleranceHigh){

inRange = TRUE; //true

}

return inRange;

}

//determine if esplora is stationary (1 = true, 0 = false)

isStill = close\_to(0.4, 1.05, magnitude);

if(t%reducedOutput >= 0 && t%reducedOutput < 3){

printf(".");

fflush(stdout);

}

}

// Magnitude was not in range of ~1, so it broke out of still loop

printf("\n\n Help me! I'm falling!");

startFall\_t = t;

double previousMag = magnitude;

double vOld = 0.0, vNew = 0.0;

double tOld = t, tNew = t;

double dOld = 0.0, dNew = 0.0;

while(1){

scanf("%d, %lf, %lf, %lf",

&t, &ax, &ay, &az);

tNew = t;

magnitude = mag(ax, ay, az);

vNew = calculateVelocity(vOld, magnitude, tOld, tNew);

dNew = calculateDist(vNew, dOld, tOld, tNew);

vOld = vNew;

tOld = tNew;

dOld = dNew;

//if there is a severe sudden change in magnitude

if(sqrt((previousMag-magnitude)\*(previousMag-magnitude))>0.4)

{

break;

}

//determine if esplora is stationary (1 = true, 0 = false)

//isStill = close\_to(0.4, 1.05, magnitude);

if(t%reducedOutput >= 0 && t%reducedOutput < 3){

printf("!");

}

previousMag = magnitude;

}

endFall\_t = t;

double timeFall = (double)(endFall\_t - startFall\_t)/1000;

double dNoResist = calculateDistance(timeFall);

printf("\n\n Ouch! I fell %.3lf meters in %.3lf seconds.", dNoResist,timeFall);

printf("\n\n Compensating for air resistance, the fall was %.3lf meters.", dNew);

int percentDif = sqrt((dNoResist - dNew)\*(dNoResist - dNew))/dNoResist\*100;

printf("\n\n This is %d%% less than computed before.", percentDif);

fflush(stdout);

return 0;

}

////////////////////////////////////////////

//Our functions.

double calculateDistance(double timeFall){

double toReturn;

toReturn = (1.0/2) \* GRAV \* timeFall \* timeFall;

return toReturn;

}

double calculateVelocity(double vOld,double magnitude, int tOld, int tNew){

double toReturn;

toReturn = vOld + (GRAV - magnitude)\*((double)(tNew - tOld)/1000);

return toReturn;

}

double calculateDist(double vNew,double dOld, int tOld, int tNew){

double toReturn;

toReturn = dOld + vNew\*((double)(tNew - tOld)/1000);

return toReturn;

}

/\* Put your lab 4 functions here, as well as your new function close\_to \*/

double mag(double ax, double ay, double az){

double magnitude = sqrt((ax\*ax)+(ay\*ay)+(az\*az));\

return magnitude;

}

//determines if value is within tolerance of point

int close\_to(double tolerance, double point, double value){

int inRange = 0;//default to false

double toleranceLow = point - tolerance;//minimum allowed

double toleranceHigh = point + tolerance;//maximum allowed

/\*printf("Is %6.4lf between %6.4lf and %6.4lf ?\n",

value, toleranceLow, toleranceHigh);//debug\*/

if(value >= toleranceLow && value <= toleranceHigh){

inRange = TRUE; //true

}

return inRange;

}

endFall\_t = t;

double timeFall = (double)(endFall\_t - startFall\_t)/1000;

double dNoResist = calculateDistance(timeFall);

printf("\n\n Ouch! I fell %.3lf meters in %.3lf seconds.", dNoResist,timeFall);

printf("\n\n Compensating for air resistance, the fall was %.3lf meters.", dNew);

int percentDif = sqrt((dNoResist - dNew)\*(dNoResist - dNew))/dNoResist\*100;

printf("\n\n This is %d%% less than computed before.", percentDif);

fflush(stdout);

return 0;

}

////////////////////////////////////////////

//Our functions.

double calculateDistance(double timeFall){

double toReturn;

toReturn = (1.0/2) \* GRAV \* timeFall \* timeFall;

return toReturn;

}

double calculateVelocity(double vOld,double magnitude, int tOld, int tNew){

double toReturn;

toReturn = vOld + (GRAV - magnitude)\*((double)(tNew - tOld)/1000);

return toReturn;

}

double calculateDist(double vNew,double dOld, int tOld, int tNew){

double toReturn;

toReturn = dOld + vNew\*((double)(tNew - tOld)/1000);

return toReturn;

}

/\* Put your lab 4 functions here, as well as your new function close\_to \*/

double mag(double ax, double ay, double az){

double magnitude = sqrt((ax\*ax)+(ay\*ay)+(az\*az));\

return magnitude;

}

//determines if value is within tolerance of point

int close\_to(double tolerance, double point, double value){

int inRange = 0;//default to false

double toleranceLow = point - tolerance;//minimum allowed

double toleranceHigh = point + tolerance;//maximum allowed

/\*printf("Is %6.4lf between %6.4lf and %6.4lf ?\n",

value, toleranceLow, toleranceHigh);//debug\*/

if(value >= toleranceLow && value <= toleranceHigh){

inRange = TRUE; //true

}

return inRange;

}

//determines if value is within tolerance of point

int close\_to(double tolerance, double point, double value){

int inRange = 0;//default to false

double toleranceLow = point - tolerance;//minimum allowed

double toleranceHigh = point + tolerance;//maximum allowed

/\*printf("Is %6.4lf between %6.4lf and %6.4lf ?\n",

value, toleranceLow, toleranceHigh);//debug\*/

if(value >= toleranceLow && value <= toleranceHigh){

inRange = TRUE; //true

}

return inRange;

}