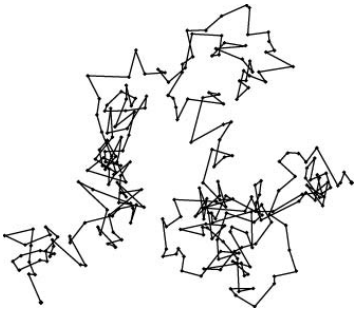
CISC 1110 (Science Section)
Brooklyn College
Professor Langsam

Assignment #5 - Brownian motion[1] (AKA The Drunken Robot)



Brownian motion (named after the Scottish botanist Robert Brown) is the seemingly random movement of particles suspended in a fluid (i.e. a liquid or gas) or the mathematical model used to describe such random movements, often called a particle theory.

The mathematical model of Brownian motion has several real-world applications. An often quoted example is stock market fluctuations.

The Roman Lucretius's scientific poem On the Nature of Things (c. 60 BC) has a remarkable description of Brownian motion of dust particles. He uses this as a proof of the existence of atoms:

> Observe what happens when sunbeams are admitted into a building and shed light on its shadowy places. You will see a multitude of tiny particles mingling in a multitude of ways... their dancing is an actual indication of underlying movements of matter that are hidden from our sight... It originates with the atoms which move of themselves [i.e. spontaneously]. Then those small compound bodies that are least removed from the impetus of the atoms are set in motion by the impact of their invisible blows and in turn cannon against slightly larger bodies. So the movement mounts up from the atoms and gradually emerges to the level of our senses, so that those bodies are in motion that we see in sunbeams, moved by blows that remain invisible.

Although the mingling motion of dust particles is caused largely by air currents, the glittering, tumbling motion of small dust particles is, indeed, caused chiefly by true Brownian dynamics.

There have been many models proposed to explain Brownian motion, however, it was Albert Einstein's (in his 1905 paper) and Marian Smoluchowski's (1906) independent research of the problem that brought the solution to the attention of physicists, and presented it as a way to indirectly confirm the existence of atoms and molecules. A simple demonstration of Einstein's explanation of Brownian motion can be seen here: http://www.aip.org/history/einstein/brownian.htm

[1] www.wikipedia.com

Imagine now a drunkard walking randomly in a city. The city is realistically infinite and arranged in a square grid, and at every intersection, the drunkard chooses one of the four possible routes (including the one he came from) with equal probability. Formally, this is a random walk on the set of all points in the plane with integer coordinates. Will the drunkard ever get back to his home from the bar? It turns out that he will.

Thought Experiment #1[2]

Think of a person who walks always with a stride of length L. If the person walks in a straight line and takes N steps, the total distance travelled is just D = N*L.

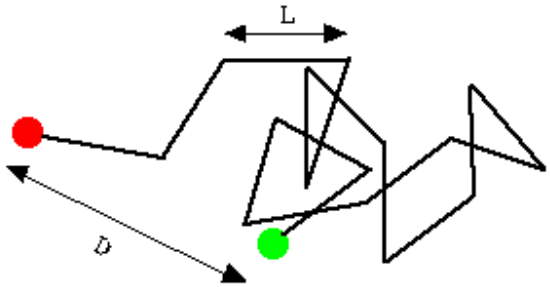
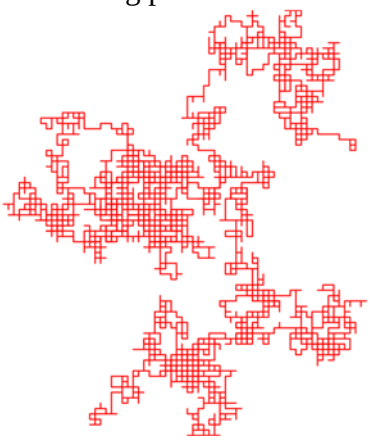


Figure 1

Thought Experiment #2

Now think of the same person heavily under the influence of alcohol The stride is still L, but the direction is now random...sometimes forwards, sometimes to the left, to the right, some staggering backwards too. In this condition, the total distance travelled is $D = \sqrt{N}L$

For example, if the distance to be travelled is 100 meters when the stride length is 1meter, the person walking in a straight line needs to take N = 100/1 = 100 steps, while the drunkard must take N = 100*100/1 = 10,000 steps. Random walk is much slower than linear walking because some of the time is spent staggering backwards. Nevertheless, the person still moves away from the starting point.



[2] http://www2.ess.ucla.edu/~jewitt/oort2-random.html3

Figure 2

Figure 3

Figures 2 and 3 represent a simulation of a random walk in two dimensions with more, and smaller, steps. In the limit, for very small steps, one obtains the Brownian motion.

PART 1

In this assignment, we will program our Lego Robot to simulate a random (drunkard) walk. You will choose a constant stride L (to be varied experimentally in the Lab) and a random turning direction. A random number can be obtained using the C++ random number function described below:

```
#include <cstdlib>
#include <ctime>

//seed the random number generator
srand(time(NULL));
x = rand();
// x will contain a random number between 0 and some
// very large number
```

Example:

This program will display random numbers between 1 and 10 on the LCD.

```
#include "robot.h"
#include "roboTask.h"
#include <cstdlib>
#include <ctime>
```

```
Lcd lcd;
Clock c;

int main()
{
    int x;

    srand(time(NULL));
    while(true){
        x = rand() % 10 +1;
        lcd.clear();
        lcd.putf("dn", x);
        lcd.disp();
        c.wait(1000);
    }
    return 0;
}
```

Use the functions that you developed in HW #3 to move forward, backward and turn (a random amount).

Repeat the sequence 50 times.

Question: The drunkard robot moves forward and backwards at random. How will you determine whether to make the power of the motors positive or negative, which will make the motors move forwards or backwards respectively?

Algorithm: If you divide any random number by 2, you will get a remainder of either 0 or 1 (why?). Thus you can write:

```
if (rand() % 2 == 0)
     //move forward
else
     //move backward
```

PART 2

Question: What happens when the drunkard hits a wall? We will assume that the drunkard, upon hitting an obstruction (after cursing) staggers backwards and then turns in a random direction.

Algorithm: In order to sense whether or not the drunkard (robot) has encountered an obstruction we will have to make use of the robot's **touch sensor**.

The NXT unit has four onboard sensor ports, labeled 1, 2, 3, and 4.

Figure 4 NXT sensor ports

Figure 5 NXT touch sensor

To read from the touch sensor, we create a touch sensor object and use the following function:

```
#include "robot.h"

TouchSensor touch1(PORT_1);    // Declares a touch
                               // sensor object on Port1
bool touched;

touched = touch1.isPressed(); // Returns either true or
                              // false to "touched"
                              // depending on whether the
                              // the touch sensor was
                              // pressed or not.
```

Example:

```
// Display "HUH" until such time as either Sensor 1 or
// Sensor 2 is activated.
// Display "OUCH" whenever sensor 2 is activated.
// Display "NICE" whenever sensor 1 is activated.
lcd.clear();
while(true) {
   if (touch2.isPressed()) {
       lcd.clear();
       lcd.putf("sn", "OUCH    ");
       lcd.disp();
       c.wait(500);
   }
   else if (touch1.isPressed()) {
       lcd.clear();
       lcd.putf("sn", "NICE    ");
       lcd.disp();
       c.wait(500);
   }
   else {
       lcd.clear();
       lcd.putf("sn", "HUH    ");
       lcd.disp();
       c.wait(500);
   }
} //end while
```

One issue still remains. How do we have the robot check to see whether a sensor has been activated while simultaneously processing the parts of the program that make our robot move? The answer is to use a programming feature known as ***multitasking***.

Multitasking:

```
#include <roboTask.h>
```

We often want to run more than one task at once. This is, unfortunately, impossible with only one processor. However, we can approximate this by switching tasks rapidly, giving each task a slice of the processor time. This is called **threading**, and each task is a **thread**.

To make use of threads in nxtOSEK, we write a standard function. These functions are the actual tasks that the robot will be performing and are labeled `task1(), task2()` and so on.

The following is a simple task:

```
task1(){
     …
     TerminateTask();
}
```

The simulator has the potential to run 10 tasks at a time. These tasks run on their own and do not have to be called. They will run until you turn the simulator or the robot off.

`roboTask.h` predefines10 dummy tasks that run simultaneously. For each task that you define in the `main,` you must be deleted the corresponding dummy task from this file (or comment it out) or it will generate a redefinition error when the program is compiled. Thus in the example below you must comment out (or delete) task1 and task2 from `roboTask.h`..

Example:

```
// Simple example of multitasking which displays output on
// the LCD while simultaneously moving the robot.

#include "robot.h"
#include "roboTask.h"
#include <ctime>
#include <cstdlib>

Motor motorB(PORT_B);
Motor motorC(PORT_C);
Lcd lcd;
Clock c;
task1() {
     motorB.setPWM(100);
     motorC.setPWM(100);
     c.wait(1000);
     motorB.setPWM(-100);
     motorC.setPWM(-100);
     c.wait(1000);
     TerminateTask();
} // end task1

task2() {
     int result;
     lcd.putf("s","HELLO");
     lcd.disp();
     c.wait(1000);
```

```
      lcd.putf("s","WORLD");
      c.wait(1000);
      lcd.putf("s","     ");
      lcd.disp();
      result = rand() % 10;
      lcd.putf("d",result);
      lcd.disp();
      c.wait(1000);
      TerminateTask();
} // end task2


// Start the program
int main() {

      // Seed the random number generator
      srand(time(NULL));

      c.sleep(10000);
      while (true);
      return EXIT_SUCCESS;
}
```

STRATEGY FOR WRITING THE PROGRAM

A Quick Reference Guide to the nxtOSEK/C++ may be found here: http://lejos-osek.sourceforge.net/html/index.html

We will be using only a small portion of the many functions described in the guide.

The following libraries should be included at the top of your programs:

```
      #include "robot.h"
      #include "roboTask.h"
      #include <cstdlib>
      #include <ctime>
```

You will be using some of the following nxtOSEK functions in your programs:

```
      Motor motorB(PORT_B);
      TouchSensor touch(PORT_1);
      lcd.clear();
      lcd.putf("s","string");
      lcd.putf("dn",integer);
      lcd.disp();
```

```
motorB.setPwM(power);
motorB.setPwM(-power);
motorB.reset();
c.sleep(time);
c.wait(time);
srand(time(NULL));
rand();
touch.isPressed();
TerminateTask();
```

1. First implement Part 1. Use the functions you developed in the previous assignment,
2. Convert the `main` of Part 1 to be a task
3. Write Part 2 as a second task.
4. Write a `main` that starts everything up and is killed along with the other two threads as in the example above, when the off button is pressed.

Test your program using the simulator in the CPlusVEBot system. Submit a printout of your program. Be sure to use meaningful variables, proper style and to comment your program as described in class.