

Lab 4
ESE 112 Fall 2007

**Everybody Dance
Now...**



1.) Introduction

In robotics, complex locomotion is an active area of research. To date not many legged robots that have even tried tackled this problem and even the best examples have often produced behaviors that are very “robotic” in nature (see http://www.plyojump.com/movies/quro/quro_fandance.wmv for probably the best example of bipedal dancing).

One of the reasons for this failure is the difficulty in chaining together dynamic movements with known outcomes and without loss of stability.

In computer science, one of the biggest failures of large-scale programming has been to create code that is modular and easy to work with. In this lab you will work to tackle both of these problems, you will choreograph a dance with RHex to music, but before starting you will have to use software engineering techniques to develop a sound, coding infrastructure.

This lab covers three laboratory sessions. During the first you will work on your software design, during the second and third you will work on the robot to coordinate with your chosen music and to make sure the movements you have chosen are stable and produce the desired behavior. On the day it is due you will present your design and dance to the rest of the class.

DANCE COMPETITION:

To add to the fun of this lab, we are going to have the dances presented in a dance competition at the end of the lab. Judging the dance competition will be two of Philadelphia’s premier ballroom dancers. This year the winner of the competition will have the opportunity to present their robot dance and a professional ballroom dancing competition in NY city.

1.) Pre-lab questions

Pre-lab Question 0: Read the appendix very carefully, what is the command needed to set up the wireless usb stick?

Pre-lab Question 1: What is static stability? Why is it important in robotics? Give an example of why static stability needs to be considered in a robotic application.

Pre-lab Question 2: What is dynamic stability? Why is it important in robotics? Give an example of why dynamic stability needs to be considered in a robotic application.

Pre-lab Question 3: Assume you have two dynamic robot behaviors; A and B. Both these behaviors have been tested and shown to work well (the behavior outputted is stable and as expected) when the robot starts from its standard starting position. Is it the case that a behavior, C, which is a sequential combination of A then B will be stable and produce the expected behavior? If not, give a concrete example where this may fail. If so, try to prove why it is guaranteed to work.

Pre-lab Question 4: What is the maximum length a function can be for this project?

2.) Lab Exercise Part 1: Code Design

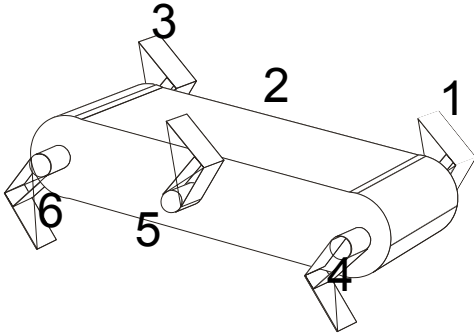
As previously mentioned one of the goals of this lab is to work on good software design. In this lab you are given a single addition (moveLeg) function (on top of calibrate and stand) that takes 4 parameters:

int legnum: The leg you want to control as diagrammed below

int direction: takes -1 or 1 for backwards or forwards

double angle: amount (radians) to move leg in direction specified in

double time: time (sec) to go from current position to position specified by angle



Note1: You may not exceed a maximum velocity for any leg of 8π radians/sec.

Note2: For coding purposes the legs are numbered from 0 to 5 as opposed to 1 to 6 shown above.

Note3: We are not responsible for lost or damaged code. Therefore it is your responsibility to save your code at the end of each session to a safe location (the easiest way to do that is to simply email your code to yourself at the end of each working session).

As you can see this one command is quite primitive and building up an entire dance (your final dance should be approx 90-120 seconds) may take hundreds if not thousands of calls to this function. In this first task you will (in groups of no more than 3 of your choice (you can't work with anyone you've worked with at any point in the semester)) come up with a software design for your dance so that no function you write has more than 15 lines of code (10 lines is preferable and you can't put two logical statements on a single line). In completing this task you will not have access to the robot, working code etc. You will need to do the design offline as is done in industry for all large software projects.

Task 1: Develop a software design based on the requirements above. Prepare a written explanation of this design (use of block diagrams, charts, written algorithms, pseudocode, etc may help) and present it to the course instructor before the start of the second lab session. In addition to the structure of the code itself this design should include the preparation of a "dance move" you plan to use and a description of how it will be coded according to your software design. Include expected values for legnum, direction, angle, and time parameters, or how they will be determined. This software design will be submitted at the beginning of the second lab session and will be included in the grading of this lab.

3.) Lab Exercise Part 2: Stable Robot Dancing

In this portion of the lab you will test your software implementation, first in simulation and then on the robot.

Note: the simulation does not run in real time so you won't be able to easily coordinate with the music in simulation.

During this portion of the lab you should pay attention to make sure that the movements you have chosen produce the behavior you expected and that the robot maintains stability at all times. (The robot should not fall over, flip over, etc (unless that's part of your routine)).

Note: This lab will be graded on your code design and development, sophistication and complexity of the dance movement and a small portion for the entertainment value of your dance.

5.) Lab Write-Up (To be done individually)

- 1.) Describe your final software design structure, similar to what was done in task 1. Include diagrams, algorithms, etc to clearly convey how your code is structured. Why did you chose this structure and what are the advantages it provides?
- 2.) Pick a move from your dance that had a relatively high risk of being unstable or one that was difficult to make stable. Describe the move (diagrams may be helpful) and discuss steps taken to make sure that move would not fail during the dance. How did your group assure stability within moves and tweak unsuccessful moves to make them more stable?
- 3.) Each dance is made from distinct "moves." How did your group decide on a sequence of dance moves? Did the robot move as you expected based on the commands you gave it? If so, how did you achieve this? If not, why and at what points would the robot fail?
- 4.) Describe your contribution to your group. As a percentage, how much of the work did you do? How much did everyone else do? Did you group work well together?
- 5.) Email your code to the joeldw@seas.upenn.edu (one email per group will suffice), just write all your names at the top of the file (in comments)). Be sure to comment your code liberally so that the reader can easily understand the purpose of your code.

Appendix: How to write code and run the robot.

This appendix is at the top of thfile in which you will code the is can be found in:

~ese112/ESE112/LABS/Lab2/SECTION/OperatorCode/Lab2.java
where SECTION is either Wednesday or Friday

LAB2 INSTRUCTIONS:

1. Write your code in this file.

All methods that you need are defined in edubotGUI class so you only have to call the following methods:

- static void calibrate()
- static void postureMode()
- static void danceMode()
- static void start()
- static void finish()
- static void moveLeg(int legNO, int direction, double absolute position, double time)

USAGE EXAMPLE: to calibrate the robot you would call edubotGUI.calibrate()

METHODS DESCRIPTIONS:

calibrate() : Calibrates the legs of the robot using the ground reference.
We call "calibrate()" in order to coordinate the legs.
Call this method before making the robot stand for the first time.

postureMode() : Call this method to make the robot stand up.

danceMode() : Call this to prepare your code to accept leg movements. It is called right after standing using posture mode

moveLeg(int legNum, int dir ,double abspos, double time)
This method moves the legs the arguments are as follows

int legNum: The number of the leg you want to move, front left is leg 0, front right is leg 3,
 back right is leg 5 etc
int dir: takes (-1,0,1), -1 is the backwards direction, 1 is the forwards direction,
 0 takes the closest path
double abspos: is the absolute position to move the leg to (you can use values from 0 to 2PI)
double time : time in seconds to get to abspos using direction dir

start() : Called after all your leg moves are inputted
finish() : Called right after start

IMPORTANT NOTE 1: Whenever you want to create your own method you should do it statically. (e.g., static void calibrate())

IMPORTANT NOTE 2: At no time can you move a leg faster than 8π rad/sec
 without the permission of the lab instructor

IMPORTANT NOTE 3: No leg can exceed 300 moves

IMPORTANT NOTE 4: There is a parameter in the file main_gui.cc called COMMTIMES when you are running
 in simulation you can keep that as low as 2, when on the robot you much change it to at least 10.

```
#define COMMTIMES          2
```

DO NOT change any other code in this file

2. Save your file.

3. Compile your file.

Type "make" from the terminal window.

Make sure that your current directory is "~ese112/ESE112/LABS/DanceLab/SECTION/"

WORKING IN THE SIMULATION ENVIRONMENT (instead of the robot)

4. In order to work in the simulation environment you must first start the supervisor by typing the following commands in a terminal shell:

commands:

```
cd ~/EduBot/Software/RobotCode/Demo/ESE112
./Demosup
```

5. In a different terminal window you need launch your code.
WARNING: you must have the joystick dongle plugged in before running you code.

commands:

```
cd ~/ese112/ESE112/LABS/DanceLab/SECTION/Operator
java edubotGUI 3000 localhost
```

edubotGUI: is the file that calls your code.
3000: is the port of the robot through which you want to hold communication
with the robot (please use only 3000 here)
localhost: indicates that we are working on the same machine.

6. Once the robot turns, stop the robot whenever you want pressing [Enter] key.

IMPORTANT NOTE: Make sure that the focus is on one of the following windows:

Robot GUI or Mode Panel.

WORKING WITH ROBOT (instead of simulation)

You will first need to insert a wireless usb stick into the computer (you can get one from on of the course staff) and then run the script setnet-rhex.

4. Open a terminal shell, start the wireless and ping the robot to make sure
that the network is up using robot_ip_address provided by the instructor.

commands:

```
ping robot_ip_address
```

5. If the network connection is present, login into the robot using secure shell.

The password will be provided by the instructor.

commands:

```
ssh root@robot\_ip\_address
```

7. While being remotely logged into the robot, use the same terminal to change

the directory where the supervisor code is located and start the supervisor.

commands:

```
cd /home/edubot/EduBot/Software/RobotCode/Demo/ESE112  
./Demosup
```

8. In a different terminal, switch to the operator working directory and run your

java program.

WARNING: you must have the joystick dongle plugged in before running you code.

commands:

```
cd ~ese112/ESE112/LABS/Lab2/SECTION/Operator  
java edubotGUI 3000 robot_ip_address
```

robot_ip_address: is the ip address of the robot provided by the instructor

edubotGUI : is the file that calls your code

3000 : is the port of the robot through which you want to hold communication with the robot

9. Once the robot turns, stop the robot whenever you want pressing [Enter] key.

IMPORTANT NOTE: Make sure that the focus is on one of the following windows:

Robot GUI or Mode Panel.

*/

```
/*SAMPLE CODE */
```

```
public class Lab2{
```

```

static final double PI = Math.PI;

//Leg Kick Move

public static void kickLeft(int pow){
    int power = 6-pow*1/2;

    if( power <= 0)
        power = 0;
    if( power > 5)
        power = 5;

    edubotGUI.moveLeg(4, 1, PI/4.0, .05*power);
    edubotGUI.moveLeg(5, 1, PI/2.0, .05*power);
    edubotGUI.moveLeg(1, 1, PI/8, .05*power);
    edubotGUI.moveLeg(2, 1, Math.PI/4, .05*power);
    edubotGUI.moveLeg(3, 0,0, .05*power);

    edubotGUI.moveLeg(4, 0, PI/4.0, .1*power);
    edubotGUI.moveLeg(5, 0, PI/2.0, .1*power);
    edubotGUI.moveLeg(1, 0, PI/8, .1*power);
    edubotGUI.moveLeg(2, 0, Math.PI/4, .1*power);
    edubotGUI.moveLeg(3, 0,0, .1*power);

    edubotGUI.moveLeg(0,1,PI, .15*power);

    //stand
    for(int i = 0; i < 6; i++)
        edubotGUI.moveLeg(i,-1,0,1);
}

public static void Main() {
    edubotGUI.calibrate();
    edubotGUI.postureMode();
    edubotGUI.danceMode();

    kickleft(1);

    //breakDanceSpin(.5);
    edubotGUI.start();
}

```

```
edubotGUI.finish();
```

```
    }
```

```
}
```

