

**University of Pennsylvania**  
**Electrical & Systems Engineering Undergraduate Laboratories**

**ESE 112: Introduction to Electrical & Systems Engineering**

**Lab 4: EduBot Dance**

Sansern Somboonsong  
Edited by Diana Palsetia  
2008-2009

**Objective**

Provide students with the background to design and implement a library of functions to perform a series of complex operations with the EduBot. Students will choose music and then choreograph a dance with the EduBot Robotic platform.

**Background**

Complex locomotion is an active area of research in robotics. To date, many legged robots can only achieve simple walking motion and mostly produced behaviors that are rigid and unnatural. Human shaped robots (humanoids) are designed with more sophisticated joints allowing for motions that are more natural. For a good example of a bipedal dancing robot, see [http://www.plyojump.com/movies/qrio/qrio\\_fandance.wmv](http://www.plyojump.com/movies/qrio/qrio_fandance.wmv) One of the challenges with complex locomotion is chaining together dynamic movements with known outcomes and maintaining stability.

**EduBot**

The EduBot is a hexapedal (6-legged) robot that has recently been developed with walking behaviors modeled after a cockroach. This robot has proven to be stable and versatile in various terrains and can probably achieve many behaviors such as climbing stairs, jumping, and jogging that have been implemented on its predecessor (RHex).

EduBot walks by alternating three legs at a time in two different cycles. This motion involves moving two legs on one side of the robot and one leg on the other (ie the front and back legs on the left moves together with the right middle leg). This will alternate on each side, pushing the robot to forward and at the same time forming 3 triangular contact points on the ground keeping the robot stable. The legs attached to the hips and are numbered or indexed according to Figure 1 below.

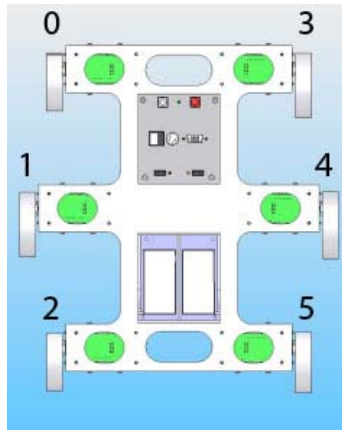


Figure 1 – EduBot Hip/Leg Numbering Convention (front of robot facing up)

The convention for the direction of rotation (-1,0,1) is determined by which direction does the leg have to rotate to push the robot forward. Imagine the robot walking forward. Looking at the legs from the left side of the robot, the legs appear to be rotating in the counter-clockwise direction. If our reference frame moves to the right side of the robot, the legs appear to be rotating in the clockwise direction. By this reason, we will consider the direction of rotation of each leg to be “1” when it will result in the motion that pushes the robot forward. The absolute angular position of the leg is based on the same reasoning. The leg is considered to be in zero position when it is vertically downward and  $\pi$  when it is vertically upward. The angle counts upward in the positive “1” direction (or the direction that will result in the robot moving forward). Figure 2 below shows the right leg angular position, looking at the leg from the right side. In other words, all the legs should point to the back of the robot at  $\pi/2$ , to the front at  $3\pi/2$ .

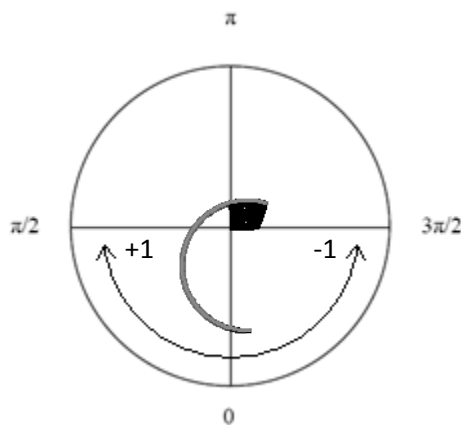


Figure 2 – Edubot Right Leg Angular Position.

The “-1” direction is the direction of rotation that cause the robot to move backwards. The “0” direction is used to command the leg to take the shortest part from its current position to the commanded position.

The EduBot measures changes in rotation by optical encoders that are attached to the shaft of every motor. The optical encoder produces thousands of electrical pulses per rotation, each of which corresponds to a small change in angular position of the leg. For the reason that the optical encoders do not give an absolute position but a relative change in position, the robot needs to be calibrated before use to have a reference point. Once that is done, the hips can be commanded to move and control the motor into some angular position (for example, when the robot stands). In reality, there is a small error between the commanded and actual position. For this reason, when you want to command the robot to move a leg around in a circle, the command must be broken up into at least two half circle movements. To make this point more clear, if we have a commanded position of “0” radians and the actual position is “6.27” radians, obviously, the difference is minimal but depending on your coding, could create unexpected behavior. To ensure that you will complete the circle, send the leg to  $\pi$  in the direction you wish it to make the circle. Then send the leg back down to 0 in the same direction as before with the same delay. This will cause the leg to make a full seamless circle.

## Coding

EduBot code base is complex and constantly evolving and will take months to be able to understand. The code needed for this lab is relatively simple and involves several functions that initialize the robot. Firstly, the project directory needs to be extracted. The archived zipped template for the dance lab project is located at:

```
~/ESE112/LABS/DanceLab Template.tar.gz2
```

Assuming you are in your home directory (`/home/ese112/`), copy the template file into your home directory and extract it by running the following command.

```
# cp ~/ESE112/LABS/DanceLabTemplate.tar.gz2 ~/
# tar -xjvf DanceLabTemplate.tar.gz2
```

A directory (or folder) called “DanceLabTemplate” is now extracted into the user’s home directory. The directory can be renamed to “myDanceLab” (or anything else) by moving the directory to a different name. For example:

```
# mv DanceLabTemplate myDanceLab
```

This directory will be referred to as the base directory of your project. In this directory, you should only write your code in the “OperatorCode/DanceLab.java” file. In the “dance” method in the “DanceLab.java” file your code where it says <write your code here> using a text editor. Of course, you should create methods of dance subroutine appropriately outside of this function. Your code should be broken up into several methods each one performing a complete dance move or part of a dance move. You will be graded on the modularity of your program so plan wisely. Do NOT change anything else in this file. The `edubotGUI` class implements a main function that calls the `dance()` function.

```
public static void dance() {
    edubotGUI.calibrate();
    edubotGUI.postureMode();
    edubotGUI.danceMode();
    <write your code here>
    edubotGUI.start();
    edubotGUI.finish();
}
```

Your code will consist of control loops, logical statements and function calls to `edubotGUI.moveLeg`. The function `moveLeg` takes 4 parameters:

```
edubotGUI.moveLeg(legnum, direction, angle, time)
```

The function parameters are described in the comments in the beginning of the `DanceLab.java` file. The `moveLeg` function can be piece together to form a dance move. Static dance move functions should be created and then called one after another to generate a dance. The dance time should add up to approximately 90-120 seconds and not legs should exceed 300 moves. If you did exceed 300 moves, the code will crash while running due to a memory problems (indicated by segmentation fault).

One simple constant that is built into the basic java libraries is the `Math.PI`. This is useful for calculating the position you want the leg to be at since it is stored as a value from 0 to  $2\pi$  (or 6.283). A good idea is to name a variable, such as "PI" and assign it the value of `Math.PI` to make your time coding easier.

Keep in mind that a time graph of each leg position is generated from the `moveLeg` function calls and sent to the robot. Accordingly, when you move one leg and want the others to be stationary at any given time, the `moveLeg` function need to be called for all legs. As an example, the following are several commands and the resulting time graphs of all of the legs. Notice that even though legs are called one after another, they all begin acting at the same time.

```
edubotGUI.moveLeg(0, 1,  $\pi/2$ , 4)
edubotGUI.moveLeg(1, 1,  $\pi$ , 2)
edubotGUI.moveLeg(5, -1,  $\pi$ , 6)
edubotGUI.moveLeg(5, 1, 0, 4)
edubotGUI.moveLeg(3, 1,  $\pi/2$ , 4)
edubotGUI.moveLeg(0, 1,  $2\pi$ , 6)
edubotGUI.moveLeg(2, 0, 0, 10)
edubotGUI.moveLeg(4, 0, 0, 10)
edubotGUI.moveLeg(1, 1, 0, 8)
```

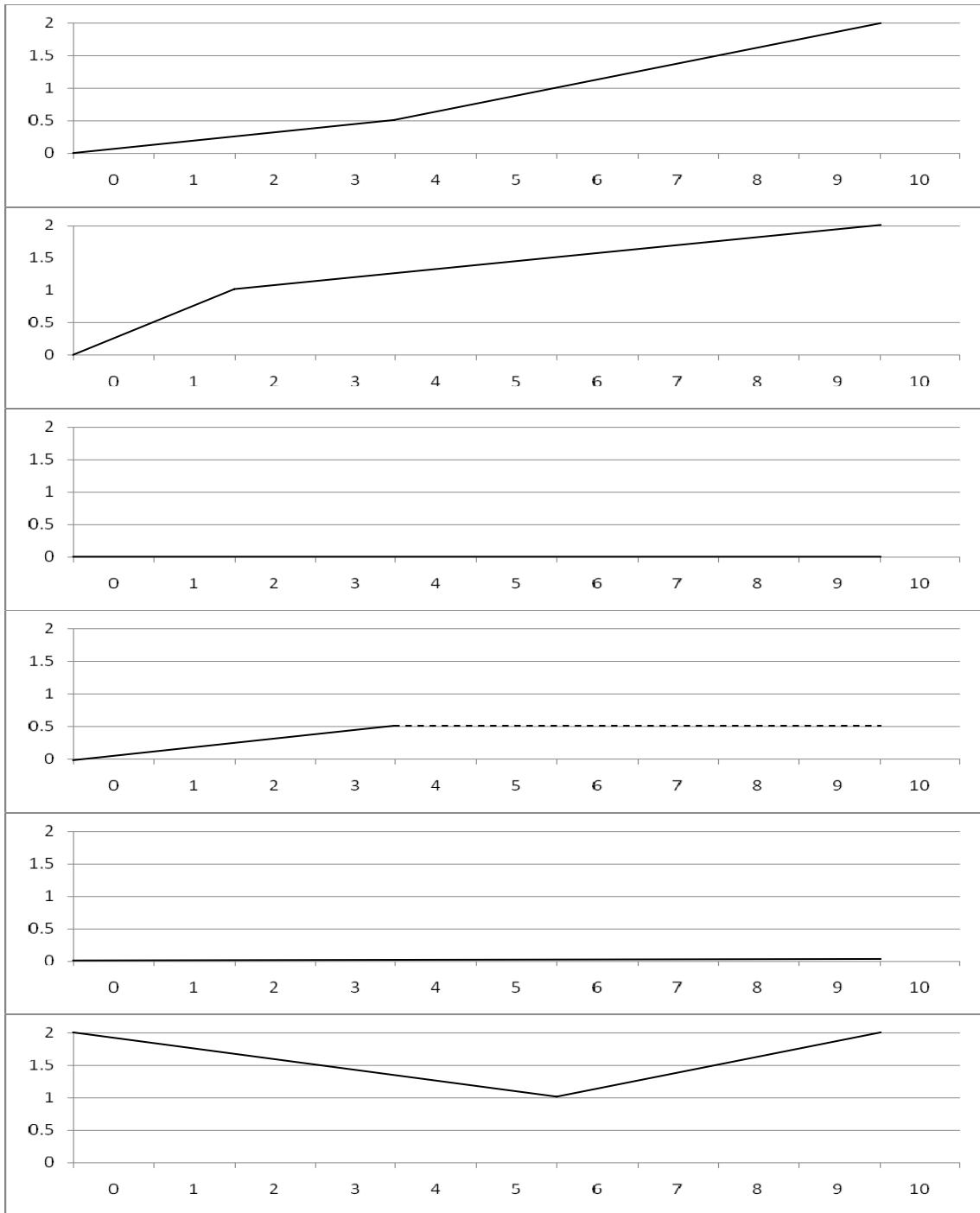


Figure 3 – Leg position over time

Attention should be paid to make sure that the movements coded produce the expected behavior and does not damage the robot in addition to maintaining stability. Note that in Figure 3, the position of leg 3 after 4 seconds is unknown since there was no command given to it. The dance routine should be consistent and reproducible. There should be a very distinct hierarchy of code, where at the top most level, each function that is executed will execute a number of sub functions that will operate the robot on a lower level. To

prevent unnecessary debugging effort, come up with appropriate names for your functions and comment your code well. After a significant amount of coding is done, it is time to test your code. Compilation should be done in the base directory of your project by running the make command.

```
# make
```

Similar to the last lab where we ran the EduBot Supervisor locally on the computer to do simulations and remotely on the robot when we want to use the robot, we can do the same with the dance lab. However, a joystick needs to be plugged into the computer in order for it to work. Since, the robot code is written in C/C++, there is a JAVA wrapper around the GUI that allows JAVA code to run almost seamlessly. This JAVA to C/C++ interface virtually presses the buttons on the joystick to control the GUI just like how the user would click on the buttons with the mouse.

To run your dance code, assuming you have the Supervisor running and your code compiled without any errors, change directory into "Operator" (not OperatorCode) and run the following command. Remember to replace <ip\_address> with localhost when running the simulation and to the robot's IP address when running it on the robot.

```
# java edubotGUI 3000 <ip_address>
```

The code is design to calibrate, get the robot in a standing position, and load the dance, you should not be clicking any buttons on the GUI until it gets to the "Dance Mode". The dance will begin when you click the "Dance Real" button.

## Material

Computer with Linux Operating System  
USB wireless adaptor  
Joystick  
EduBot

## Prelab

1. What is the command needed to set up the wireless USB adaptor and why do we need to use a wireless USB adaptor? (you may need to refer to Lab3 manual)
2. 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.
3. 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.
4. How many seconds should your dance be and what is the maximum number of moves per leg?

## Lab instructions

1. Form groups and run the sample code given in the project folder template in simulation and on the robot.
2. Choose an appropriate song to create a dance to. Your song should be rated PG13.
3. Follow the rules and guidelines laid out in this lab and come up with a few dance moves. Prepare a written explanation of this design (use of block diagrams, charts, written algorithms, pseudocode, etc) and present it to the course instructor before the start of the second lab session.
4. Test your software implementation in the simulation and ensure that it works as expected
5. Run your programmed dance on the robot and synchronize it with your music in preparation for a performance.

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.

## Questions

1. Describe your final software design structure. 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?

Submit your program to blackboard through the digital dropbox under the lab appropriate section (101/102) of Blackboard. Be sure to comment your code liberally so that the reader can easily understand the purpose of your code. Put your java file in a folder called ese112\_DanceLab\_XX\_XX\_XX (where XX is the first and last initial of a group member) and archive the contents. Finally submit ese112\_DanceLab\_XX\_XX\_XX.zip to Blackboard.