CS148
Building Intelligent Robots

Week 2 : Introduction to brickOS *Out: 10 Feb 2004*

Preliminary Tasks *before 12 Feb 2004, 9am*

The purpose of this week's lab and project is to get you acquainted with programming in brickOS. You will only be introduced to a limited set of commands in this lab, and will be expected to read through the documentation and understand the rest on your own. Complete the following before coming to lab on Thursday:

- Be fairly comfortable with programming in C and using C syntax. If you need assistance, talk to a TA.
- Install brickOS on your own computer (only one per group is necessary). You will need to follow the HOWTO / QuickStart guide, available on the web.
- download the helloworld program from your own computer (`C:\brickOS\demo\helloworld.lx`) to your RCX and run it.
- Build a robust tankbot with two treads, one motor controlling each tread, and a touch sensor. Look at the sample tankbot on the webpage for inspiration.

  (http://www.cs.brown.edu/courses/cs148/2003/handouts/simple_tankbot)

1 Setup

Pick a computer in the MS lab.

First you need to map the cs148 course directory to the 'L' drive. To do this, open up 'My Computer' and click on Tools -> Map Network Drive. Select 'L' from the Drive menu. In the folder type:

'\\maytag\course0\cs148´

and click finished.

We have configured brickOS in the MS Lab to be a little different from your dorm room installation. There should be an icon called "Cygwin" on the desktop after you log in. This is the bash shell that you want to use. Start it up.

You'll notice that when you open up Cygwin it looks and works like a shell on Linux. Right now you are in your home directory, the same home directory you would be on a Linux machine. Create a directory to work in. Then 'cd' into that directory and copy the brickOS lab files from the course directory with

```
cp -r l:/asgn/brickOS/* .
```

Some general reminds about using the Makefile.

- To download the brickOS kernel to the RCX: `make brick`
- To compile a down a program: `make install`
- The Makefile works exactly the same if its on a Windows or Linux machine.

Lab 2: Introduction to brickOS *12 Feb 2004*

1.1

You will need to know how to control the motors so that the robot will go (approximately) where you want it to go.

Review the documentation on the course web site about using motors, timing, and playing sounds.

Task: Make the robot go forward for one second, then turn approximately 180 degrees and stop. After the robot has stopped moving, make it play a short tune of your own choice. If you want, you can use the skeleton file provided in *step1.c* that you already copied from the course directory.

You may find the following commands helpful in this section:

- `motor_X_speed`
- `motor_X_dir`
- `msleep`, `sleep`
- `dsound_play`

1.2

Multitasking is a crucial ability that your robots will need to be able to do.

Review the web site documentation on task management.

The concept of threading is fairly simple. The idea is that you want to do multiple things at the same time, so to do this, you spawn(create) a new thread. When a new thread is created, it begins execution at a specified point, and runs concurrently with other threads that are currently running (including the thread that created it).

We have provided a skeleton file for you in *step2.c*. Feel free to use this or write your own.

Task: Write a multi-threaded program. Your program should do the following:

- The main thread should create two child threads and then enter an endless loop.
- The first child thread should make your robot turn counterclockwise in place for one second, then turn clockwise in place for one second, and then repeat endlessly.
- The second child thread should play a tune (at least 4 notes), then play it again and again and again... Keep a global variable that keeps track of how many times the tune has been played.

You will probably want to use the `execi` command

1.3

Much of what your robot will do is to wait for things to happen. brickOS provides a `wait_event()` command that allows you to do just that - wait for something to happen. Review the web site documentation and make sure you understand the sample files. The sample files are also located in the course directory in a directory called *samples*. This step will get you acquianted with events and handling them.

There are two steps involved in creating an event. First, you need the wakeup function. This is just a function that returns zero when your event hasn't happened yet, and a non-zero value when your event has happened. For example:

```
wakeup_t dkey_pressed(wakeup_t data);
```

This function will return non-zero only when a key has been pressed. To wait for this event, we would use the command `wait_event(dkey_pressed, 0);`

One thing to note about using `wait_event` is that it always takes a second parameter of type `wakeup_t`, but it almost never means anything. In almost all cases, you can safely use 0 as the value you pass to the `wait_event` function. The data value you pass it is passed to the wakeup function, but few of them actually use it.

Task: Modify your program from part two so that it does the following.

- The main thread should once again create two child threads. The two threads should do the same thing as before (turn back and forth endlessly, play a tune endlessly).
- Instead of having the main thread enter an endless loop, however, it should wait until the tune has been played 5 times. After the tune has been played five times, the main thread should kill the two child threads and then exit. You must use `wait_event` for this part. (hint: write a `wakeup_t` function that returns true when the tune has been played enough times)

1.4

For the final step, you will learn how to use a touch sensor. The touch sensor is the simplest of all sensors. Other, more advanced, sensors will be introduced next week.

You can read a touch sensor by using the macro `TOUCH_`*`X`*, where *`X`* is the number of the input port to which your touch sensor is attached. This macro will evaluate to 1 if the touch sensor is pressed down, and 0 otherwise.

Task: Modify your program from part three so that it doesn't start until you've pressed the touch sensor.

Project 2: Introduction to brickOS *due 17 Feb 2004, in class*

Specification: The exercises in this lab have covered some of the more important aspects of brickOS programming. As the semester progresses, you will be expected to learn the rest of what you need on your own. The documentation for everything is provided on the course website and course directory, and feel free to ask a TA for help if you need it.

For this week's project, you will experiment with both of your touch sensors, and implement a simple obstacle-avoiding robot.

The most basic robot you can use for this project is a tankbot. However, you will be awarded 5 points for building a robot with a more advanced drive system. The following web site provides examples of different drive mechanisms:

http://www.cs.dartmouth.edu/~robotlab/robotlab/courses/cs54-2001s/locomotion.html

Build a wheeled or tracked robot, not a legged robot. Note that the tankbot uses skid-steer drive, so this system is not considered a "more advanced" drive system. Some of these drives are easier implemented with legos than others, so choose wisely. (This web site provides a good outline of the pros and cons of each drive system.)

Your robot will be placed in the center of an "arena," surrounded by several layers of obstacles such as walls and boxes of different shapes. You will not know the placement of the obstacles beforehand, so it will be impossible to simply program the robot to avoid without using the sensors. In addition, your robot should be able to find it's way out of the enclosure of obstacles more than once (we should be able to pick it up after it clears the obstacles, put it back in the center facing a different direction, and it should be able to find its way out again). Finally, your robot should assume that it has cleared all the obstacles if neither bump sensor has been activated for 15 seconds, and the program should end (motors off).

- How are you going to use your sensors? Do you want two front sensors that can differentiate between left and right, or a front and rear sensor? How will this scheme work with your chosen drive system?
- You should make good use of abstractions when coding the turnRight() or turnLeft() functions (called when the bumpers are toggled).
- Consider how the robot is going to account for corners—this is tricky, and very important.

Paper Handin: Be sure to discuss the following items in your writeup:

- Briefly explain the design of your car and drive mechanism. If you based the design on material you found on a web site, in a book, etc., cite those sources.
- Explain in detail your choices for bumpers, and the underlying code.

- Give a general overview of the flow of control of your program. Don't forget to mention any subroutines or tasks that run simultaneously. (Pseudocode is acceptable, but not required.)
- Be sure to include explanations of any nontrivial code, or special procedures.
- Include a diagram of your robot.
- Keep it brief! The text portion of your writeup should be no more than two pages.

Grading: You will be graded on equally on the functionality of your robot (50%) and on your report (50%). The robot will be graded according to how effectively it circumnavigates the obstacles. For example, it is a better design choice to make a robot that hits the obstacles more times but has a better idea of the size/shape of the obstacle than it is to make a robot that, after hitting the obstacle once, attempts to move around it without any idea of it's size/shape. In addition, your robot should be able to find it's way out of the enclosure of obstacles more than once.

You should handin your code for the robot by running

/course/cs148/bin/cs148_handin brickOS from the directory containing your source code.

Robot	
Obstacle avoidance	30%
Bumper design	10%
Drive mechanism	10%
Report	50%