Research Lab 1: Embedded Systems

Part I – Bringing up leJOS on your PC

A concise definition of an embedded system is given by the following Wikipedia entry, "An **embedded system** is a computer **system** with a dedicated function within a larger mechanical or electrical **system**, often with real-time computing constraints." Thinking about this definition for a moment, it becomes clear that virtually all modern consumer electronics are comprised of embedded systems, not to mention their traditional application in industrial systems and control. So it should come as no surprise that the topic of embedded systems should figure prominently in a design course, given their importance as building blocks in modern systems and devices. The objective of Research Lab 1 is to become familiar with embedded systems in the form of the Lego Mindstorms design kit, given that the final project will be based on this hardware and software.

The heart of Mindstorms is the Lego "Brick", which serves as a general purpose embedded controller with 4 sensor inputs and 4 motor outputs, operated by a simple user interface (UI) consisting of an LCD display and a set of push buttons. Inside the brick is a Texas Instruments Sitara AM1808 microcontroller, which is based on the ARM 926EJ-S core running at 300 Mhz – a bit of a mouthful, but essentially the same technology that ran the first generation of smartphones. Devices such as the AM1808 are often referred to as a SOC, system on a chip, and contain most of the key components of a computer system on a single silicon die. You will learn more about these devices in later courses such as Microprocessors and Embedded Systems. For now, you can think of microcontrollers as a scaled down version of what you would find in a typical laptop computer.

What turns a microcontroller into an embedded system is the software that provides the required functionality, thus a large part of the process involves the design, implementation and testing of software. A key element in this process is the *development environment* that provides the basic tools for code development, testing and documentation. Having taken ECSE 202, you should already be familiar with development environments similar to the one we will be using, Eclipse, and the Java programming language. The reason for choosing Java is that it has all the constructs necessary to support embedded systems development (in fact, it was designed especially for this purpose), and that the complexity and detail of low-level operations (such as motor control) can be hidden through class abstractions. This makes it possible to put together fairly sophisticated systems using existing class libraries, while still permitting a high degree of customization using Java's inheritance mechanism, for example.

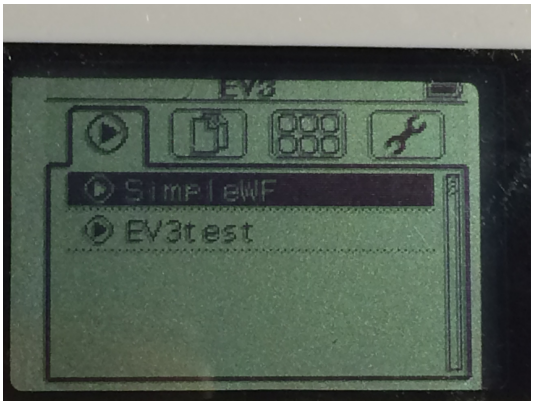
Of course the default Mindstorms setup does not support Java, so we need to provide alternative firmware for the Brick. This turns out to be quite easy to accomplish as the Brick runs the Mindstorms environment on top of Linux. The micro SD card[1] included in your kits contains a different version of Linux with the leJOS EV3 environment as the replacement user interface. To switch from Mindstorms to leJOS, all you need to do is insert the SD card and reboot the

[1] Instructions for generating an SD card for the EV3 can be found under the Link to Resources section of the course home page.

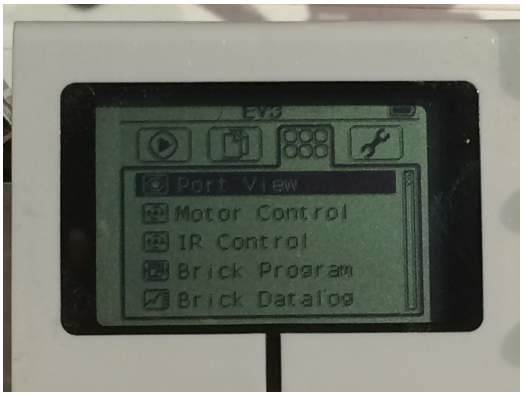
Brick. Before proceeding further, read the EV3 User Guide,
http://www.cim.mcgill.ca/~ferrie/dpm/ev3-user-guide-EN.pdf,
and familiarize yourself with the Mindstorms system, paying particular attention to the Brick and its functionality. Once you have completed this, proceed to the checkout procedure.

Checkout Procedure:

Each kit comes with a rechargeable battery pack. Before proceeding further, make sure that your packs are fully charged, and that there is no card inserted in the SD socket. Powering on the brick (button at center of array) will bring up the Mindstorms software environment. Although we will *not* be using Mindstorms for the course, it useful to ensure that your hardware is operating properly. If all is OK, you should see something like shown in the figure below.



Using the right arrow key (rightmost in the array), click two positions over to the utilities menu. You should see the display below. Of immediate interest are the Port View and Motor Control



utilities. Your kits are equipped with a touch sensor, light (color) sensor, ultrasonic rangefinder, inertial measurement unit, and a sound sensor as well as 3 motors. Sensors plug into Ports 1-4 and motors into Ports A-D. Using the Port View utility, you can experiment with each sensor to see how it behaves as well as verify correct operation. Similarly the Motor Control utility will allow you to vary the speed and direction of each motor. In fact, you should write down your observations and use these as a baseline reference. For example, if you suspect that a motor or sensor is malfunctioning, then a check with the appropriate utility and a comparison against your

baseline values will quickly resolve any issues. In the event that a component is malfunctioning, you may exchange it for a replacement at the Parts Counter on the 4th floor of the Trottier building. Once you have validated your kit you may proceed to installing leJOS.

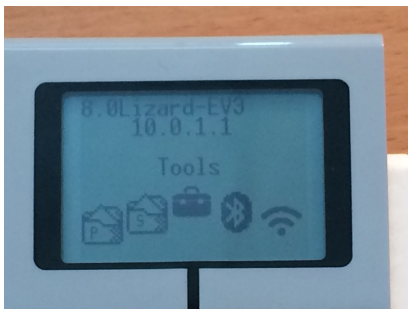
leJOS Installation:

Mindstorms provides a basic environment for designing mechanisms but limits access to the more powerful features of the sensors and actuators, and makes it somewhat awkward to exercise fine control. For this reason we use leJOS, which consists of a JVM (Java Virtual Machine) and a set of classes that provide a fairly rich environment for implementing embedded robotic systems. Software development uses already familiar tools, namely the Eclipse IDE (Integrated Development Environment) with some add-ons to facilitate packaging code for the Brick's JRE (Java Runtime Environment) as well as communicating with it.

To install the leJOS runtime environment, insert the microSD card provided with your kit into the SD card socket on the side of the Brick (with the power off). Restart the Brick by pressing the center button. Note that the first time you do this, the system will take several minutes to respond as it must first complete the installation of leJOS on the SD card. Subsequent restarts will be much faster. If for some reason you are unable to bring up leJOS on your Brick, please see one of your Lab TAs or else return your microSD card for replacement. Please do not attempt to reload the SD card with the code from the leJOS site (www.leJOS.org) as the cards provided to you may include local changes that are not reflected in the leJOS distribution.

If all is well you should be greeted with the following on the EV3 display:

Depending on your platform (Windows/Mac/Linux), the procedure for installing the support environment on your PC varies. Of particular importance are the numbers below the top line of the display, 10.0.1.1 in this case. These correspond to the IP address of the Brick when plugged into your PC with the USB cable (using Ethernet over USB). You will need to type these in later to PC side applications that connect to the Brick. For now you can verify correct operation of leJOS by clicking the right button exactly 3 times, bringing you to the Tools menu:



Clicking the center button to select Tools brings up the available choices in the following sub-menu:

As before, you can verify correct operation of your sensors and motors using the Test Sensors and Remote Control utilities respectively. With leJOS up and running on the Brick, the next step is to install the development environment on your PC (Windows/Mac/Linux are supported).

Java:

Java does not execute directly on the hardware of a particular machine; it runs within the Java Virtual Machine (JVM). The JVM provides the abstraction layer for Java that hides the details of the hardware and software environments of the platform on which it is run. Together, the JVM and platform specific classes are referred to as the Java Runtime Environment (JRE). Whereas Java source code is indeed, "Write once, run everywhere", the associated JRE is platform specific. For an embedded systems developer, i.e. *you,* this can cause a lot of grief as it is more than likely that you will be developing software in an environment that is different from the one that it will be run on.

Host Preparation:

Java is available for all major platforms at the following link: https://java.com/en/download/ The server will figure the host environment out from your browser and download the appropriate JRE package to your computer. This gets installed the same way you would install any other software package, and usually includes a test program to make sure that Java is installed correctly. It is more than likely that Java is already installed, but you might want to make sure

that your installation is up to date[2]. For this course we strongly recommend that you use the Eclipse IDE as there are plugins available that directly support leJOS. The latest version of Eclipse is available at https://eclipse.org/downloads/, but here there are some choices to make – the version and whether to download the 32-bit or 64-bit version. Choose Eclipse IDE for Java Developers, 32-bit version. The reason for choosing the 32-bit version is that the leJOS plugin (which you will install a bit later) appears to have some 32-bit dependencies with the current version of leJOS. Until these get resolved, the 32-bit version is the one to use.

Eclipse itself is downloaded as a .zip archive, which means you will need a utility such as WinZip to extract the content. Unlike native applications, which reside in specific locations, Eclipse is written as a Java application so it can be placed anywhere convenient. However, it's usually a good idea to place it where your other applications reside (C:\Program Files (x86)\eclipse, Windows; /Applications/eclipse, OS X; /usr/local/bin/eclipse, Linux).

leJOS Package:

As with Eclipse, leJOS is a package that runs on top of Java, and is available at the following URLs:

1. Windows: http://sourceforge.net/projects/ev3.lejos.p/files/0.9.0-beta/leJOS_EV3_0.9.0-beta_win32_setup.exe/download
2. Linux/Mac: http://sourceforge.net/projects/ev3.lejos.p/files/0.9.0-beta/leJOS_EV3_0.9.0-beta.tar.gz/download

The Windows setup is straightforward, simply run the installer as per the usual software install and answer a couple of questions. First, make sure that the Java Development Kit version selected is 1.7.X and not 1.8.X. The current version of leJOS has not yet been validated with JDK 1.8. Next, make sure that the install location is consistent with your version of Windows. The default will be C:\Program Files\leJOS EV3. However if you are running a 64-bit version of windows (which is more likely), you should select C:\Program Files (x86)\leJOS EV3. Ultimately, this does not make any difference as far as Java is concerned, but it is a good idea to follow the Windows conventions anyway with respect to 32-bit and 64-bit code. Finally, you will also have the opportunity to select Additional Sources in the form of Sample code and even the source code for the leJOS EV3 Development Kit. You can select these at your leisure (hint: you can often learn a lot more by looking at working source code than restricting yourself to a language manual).

The Linux and Mac (Unix) worlds usually require a bit more work. From the link cited above, download the file named leJOS_EV3_<version>.tar.gz and unpack (if this does not happen automatically, double-clicking on the .gz file will usually do the trick). As with the Eclipse installation, you are free to chose where to place the resulting folder. However, it is generally a good idea to place it with your other applications as discussed above for Eclipse. In any case, make note of this location as it will be needed shortly in the Eclipse configuration.

[2] Warning! Some older Java applications require that particular JRE's are installed. This is normally handled correctly by the Java support system on your particular machine, but sometimes does break (e.g. on OS X with the switch from JRE 6 to 7).

Eclipse Configuration:

If running on a Windows system, a shortcut to the Eclipse application will appear on the Desktop and/or Taskbar. For Mac or Linux users, you may need to create a shortcut (i.e. alias/link) to the application contained in the Eclipse folder. The next step is to launch Eclipse and install the leJOS plugin as follows. Navigate to the Help menu and select Install New Software; this will bring up the Software Installation pane. In the Work with window type the following URL: http://lejos.sourceforge.net/tools/eclipse/plugin/ev3. This should bring up "leJOS EV3 Support" in the Name window; click on the Select All button followed by Next to complete the installation. If successful, you should see a new menu installed called leJOS EV3. Finally, we need to configure the plugin by telling it where the leJOS development tools live as well as the IP address corresponding to your brick. If on a Windows system, navigate to the Windows menu and select Preferences and then leJOS EV3, otherwise the Preferences pane should appear under the Eclipse menu (Mac/Linux). Fill in the EV3_Home field with the location where you installed the leJOS folder, e.g., C:\Program Files (x86)\leJOS E3 or /Applications/leJOS EV3 or /usr/local/bin/leJOS EV3. It is suggested that you enable (checkbox) Run Tools in separate JVM and Use ssh and scp. You probably also want to enable Connect to named brick and fill in the IP address for your brick. This will be used to automatically upload program images directly from Eclipse. The remaining checkbox, Run program after upload, is entirely up to you.

Communicating with the Brick:

There are two main methods available to communicate with your brick, USB cable and Wifi connection. Bluetooth is another option, but can be problematic on certain machines so we'll ignore this option for now. In both cases, the Internet Protocal (IP) is used to communicate with the Brick, the only difference being the physical connection to the host computer (i.e. your PC). Using the USB cable is the simplest approach as it doesn't depend on the availability of a nearby Access Point (in fact, the APs at McGill are configured to prohibit local connections of this sort). On the other hand if you are at home, this should not be a problem. If you have successfully connected your smartphone to your home network, then the procedure is similar using the leJOS menus.

If using the USB cable to connect, use the IP address 10.0.1.1 as the "Name" of the brick in the leJOS EV3 configuration menu. This appears on the second line of the display under the name of your particular brick. If connected via Wifi, the third line of the display (under 10.0.1.1) will contain the IP address returned by the router. Use this address instead of 10.0.1.1 to communicate via Wifi.

In any event, it is a good idea to connect via USB first, and then use the Start EV3Control program under the leJOS EV3 menu in Eclipse. This will enable you to change the Brick configuration interactively and bypass the leJOS menu system (especially useful for typing in Wifi passwords).

Hello World:

The final step in the checkout procedure is to create a simple Hello World program in eclipse, upload it to the brick, and confirm proper execution. Since we are creating this program to run in the leJOS environment, we need to make sure the proper libraries are included in the build. Fortunately Eclipse makes this process rather painless as follows:

1. Create a new project: File>New>Java Project

The minimum required is a project name, in this case I chose HelloWorld; the remainder of the defaults are sufficient.

2. Convert to an EV3 project: Right Click on the project name in the Project Explorer pane >leJOS EV3>Convert to leJOS EV3 project

You now have the skeleton of your project.

3. Create a new package: File>New>Package

Generally good to define a unique package name.

4. Create a new class: File>New>Class

The minimum required is the class name, here I chose HelloWorld and left the defaults intact.

5. Under HelloWorld>src>myPackage>

In the Project Explorer you should find HelloWorld.java. Click on this entry to bring up the source file created by Eclipse. It should look as follows:

```
package myPackage;
public class HelloWorld {

}
```

Now fill in the template with a simple Hello World program:

```
package myPackage;
import lejos.hardware.*;

public class HelloWorld {
	public static void main(String[] args) {
		System.out.println("Hello World");
		Button.waitForAnyPress();
	}
}
```

6. Upload and run the program: Run

Depending on the checkbox setting in the leJOS EV3 preferences, the program will upload or upload and execute. If the program appears to be stuck in an infinite upload, go back to the preferences pane and uncheck Run program after upload. Upload again using the Run button and launch the program from the EV3 front panel. A nice alternative is to use the EV3Control program to launch and run programs with the output appearing on the remote console. You can access EV3Control from the desktop using leJOS EV3>Start EV3Control.

For Mac/Linux users, you might need to manually initialize usb networking on your machine. It will appear as a network interface called RNDIS/Ethernet Gadget which you should configure manually with the following parameters: IP Address – 10.0.1.X (X<-[0,255]); Subnet Mask – 255.255.255.0; Router – 10.0.1.1. This should be sufficient to bring up the usb interface. If you are still unsuccessful, please see one of the TAs.

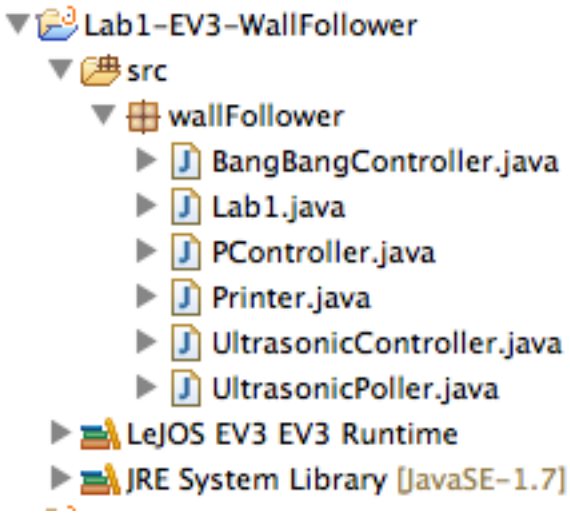
Part II – Implementing the Wall Follower

For this part of the lab you need to construct a two-wheeled robot equipped with an ultrasonic sensor; you may use any of the designs on the DPM myCourses page under Robot Designs. The theory behind the wall follower robot is presented in the course notes and discussed in the class lectures. There is also some code that implements the wall follower, but is provided more to give you some examples of how to use the different leJOS classes.

Rather than rolling your own code from scratch, you will be provided with a mostly completed Java project, Lab1-EV3-WallFollower-code.zip, which you can easily import into Eclipse in the following steps.

1. Create a new Java Project called Lab1-EV3-WallFollower
2. Right-click on Lab1-EV3-WallFollower and select leJOS EV3>Convert to leJOS Ev3 project
3. Create a new Java Package called Lab1-EV3-WallFollower
4. File>Import>Archive File>Next
5. Navigate to wherever you downloaded Lab1-EV3-WallFollower-template.zip (you do this from inside myCourses) to and select this zip file as input. Next.

If you now examine the project you just created in the Project Explorer window,
It should look as follows:



If you are new to Java (and that includes all the COMP 202 folks that might not have been paying attention :), this code looks way different from the example done in class. It is written more in an object-oriented style, showing the different Java constructs, and is intended to get you thinking more Java-like. The actually coding you need to do is minimal, restricted to the BangBangController and PController classes (search for // todo). If you understand the example in the class notes, then you should be able to figure out how to fill in the missing parts for the BangBangController class.

Since these are *research labs*, leading up to the final project, there is usually something missing that you need to look up (Google is your friend) or figure out for yourself. Here that something corresponds to *proportional control*, where the correction applied is proportional to the error as opposed to a constant in Bang-bang. For both types of controller, the code you need to write is relatively small. The exercise here is to understand how the code provided to you works well enough to provide the missing pieces.

Code Details:

For the purpose of making a wall following robot, it is necessary to develop a good means of reading the ultrasonic sensor. The LEGO Mindstorms ultrasonic sensor is capable of getting data at around 25 times per second, however due to the fact that we won't be using timers, we will poll as fast as the ultrasonic will allow in a thread. After each polling, the UltrasonicPoller will update the value of the distance integer in the controller for which is being processed at that time.

In Lab1.java we create all the objects that we will need, which includes the two controller types to be implemented. Based on whether the left or right button on the NXT is pressed, it will choose the controller to be run accordingly.

The UltrasonicPoller object (called usPoller) is what gets the data from the ultrasonic sensor, and gives it to any object of a class that implements the UltrasonicController interface. Its constructor thus takes as arguments an UltrasonicSensor object (which is part of the leJOS API), and any object implementing the UltrasonicController interface. In the UltrasonicPoller.java file provided, we see on line 19 how it gets the ultrasonic data, then passes it off to the UltrasonicController, cont:

```
public void run() {
          int distance;
          while (true) {
               us.fetchSample(usData,0);             // acquire data
               distance=(int)(usData[0]*100.0);      // extract …
               cont.processUSData(distance);         // now take …
               try { Thread.sleep(50); } catch(Exception e){}
          }
     }
```

This is where any filters for the ultrasonic sensor should be implemented, such as the removal of spurious 255 values.

The final element is the processUSData(int distance) method in Pcontroller and BangBangController.

```
public void processUSData(int distance) {
	this.distance = distance;
	// TODO: process a movement based on the us distance passed in
	// (BANG- BANG style)
	// wall on the right
}
```

With the preceding as background, the formal laboratory description begins on the following page. All of the labs in this course will conform to this format.

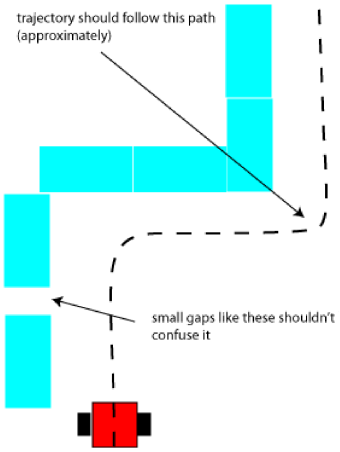
Objective

To navigate around a sequence of cinderblocks, making up a 'wall' containing gaps and both concave and convex corners, without touching it or deviating too far from it.

Method

See the attached code for a working example of how to do wall following. Modify it to:

- Avoid getting confused by gaps
- Turn concave corners (i.e. corners which the robot would run into were it to travel in a straight line)
- Turn convex corners sharper
- You need to implement both the bang-bang and P-type controllers, each is contained in their own .java files. Build the "Stronger with Rotating US" robot, whose LXF file is provided on WebCT. You may modify the ultrasonic sensor mount as you please. (Note: you will need to download the Lego Digital Designer software available at http://ldd.lego.com/en-us/download/, available for Mac and PC platforms, to view .lxf files).



Data

All data for this lab is qualitative.

Data Analysis

a) Did the bang-bang controller keep the robot at a distance bandCentre from the wall? Why is it expected that the robot will repeatedly oscillate from one side of the band to the other with the bang-bang and p-type controllers?

Observations and Conclusion

What errors did the ultrasonic experience? Were these errors filterable? Does the ultrasonic sensor produce false positives (i.e. the detection of non-existent object), false negatives (i.e. the failure to detect objects), or both?

Further Improvements

What improvements could you make to both the physical or software designs to improve performance of the wall follower? (At least 3 are needed) Are there any other controller types that may have better outcomes than the bang-bang and p- type?

To Submit

• One document in .pdf format containing the lab report